zhoukeli2005的ChinaUnix博客

暂无签名

zhoukeli2005

博客访问： 13074
博文数量： 1
博客积分： 0
博客等级： 民兵
技术积分： 37
用 户 组： 普通用户
注册时间： 2013-02-08 15:53

加关注 短消息
论坛 加好友

我的朋友

最近访客

 hymyg
 whutchao
 zxy87729
 milkywat
 reb00t
 huyi91
 asia318
 seshenli
 CUTianru

Python 学习笔记 02 – List 推导式

2013-02-08 17:03:27

分类： Python/Ruby

by周克利

(基于Python 2.7.3)

Python语言以简易明了著称，但初次学习Python，却被很多语法搞的昏头涨脑。List comprehension绝对是其中之一。

一、困惑

问题一：列表推导式中的变量，是全局变量？还是局部变量？还是闭包变量？

注：一个简单的列表推导式，如下：

```
1 a = [ x for x in range(10) ]
```

这里的变量x，是局部变量吗？在列表推导式结束后，还能访问变量x吗？

问题二：列表推导式，推导过程到底是从左往右？还是从右往左？

注：一个简单的列表推导式，如下：

```
1 a = [ (x, y) for x in range(2) for y in range(3) ]
```

如果写成伪码，是：

```
1 for x in range(10):
2     for y in range(10:
3         a.append((x, y))
```

还是：

```
1 for y in range(10):
2     for x in range(10):
3         a.append((x, y))
```

虽然这个问题看起来很奇葩，但让我很困惑。

问题三：列表推导式中，for语句和if语句之间的关系是什么呢？

注：不光是for语句和if语句之间的关系，多个for语句之间，多个if语句之间，for语句和if语句之间，它们的关系又是什么样的呢？

问题四：列表推导式，到底是怎么运行的呢？

注：虽然看到列表推导式，凭借猜测，也能猜个八九不离十。但总不知道List推导式具体如何运行的，心中总不踏实。

二、释惑

关于List comprehension，有着千言万语的困惑：当里面夹杂着多个for、if语句时，如何读懂？更要命的是：如何写List comprehension才是正确的呢？

关于这种种疑问，归根到底，是不了解Python是如何处理List comprehension的。

1. 关于GET_ITER、FOR_ITER字节码(opcode)

Python中有迭代的概念，最常用的要数for循环了。一个简单的for循环如下：

```
1:      for x in A:
2:
3:              do_something
4:
5:
```

对于这个for循环，Python会大致生成如下字节码：

```
1:      iter = GET_ITER( A )
2:
3:      x = FOR_ITER(iter)
4:
5:      if not x : jump to 6
6:
7:      do_something…
8:
9:      jump to 2
10:
11:     (The End)
```

在这里面，起到重要作用的，就是GET_ITER、FOR_ITER这2个字节码。

其中，GET_ITER是取出一个Object的迭代器（调用PyObject_GetIter()，如果是一个类的对象，调用其__iter__方法）；

之后，就会不断对这个迭代器执行FOR_ITER指令（如果是一个类的对象，调用其next方法）；

如果FOR_ITER指令迭代不到下一项了（通常是遇到StopIteration异常了），就跳出for循环，否则会一直迭代下去。

2. 关于POP_JUMP_IF_FALSE指令

这个指令比较长，第一次见有些被吓到，不知是干什么的。

但是仔细一看，这个指令还是比较好理解的：

首先，Python的每个函数都有自己的运行栈，所有的临时运算结果都要放在这个栈上的，例如以下简单代码：

```
1:      if a > 0:
2:
3:              do_something
```

这里的变量a，如果是个全局变量，那么它存在全局变量Dictionary里；如果它是个局部变量，那么它存放在局部变量数组里；如果它是个闭包变量，那么它存放在闭包变量数组里，总之，它是不在函数运行栈里的。

但是，变量a和常量0的逻辑比较的结果，是一个临时的运算结果，这个运算结果是要放在函数运行栈里的，以方便后面if判断时使用。

那么，Python对这么个简单的if代码，会大致生成以下字节码：

```
1:      进行a > 0判断，
2:
3:      Push(结果)
4:
5:      POP_JUMP_IF_FALSE
6:
7:      do_something…
8:
9:      (The end)
```

Python会在逻辑运算后，将逻辑运算的结果自动Push进函数的运行栈内。那么，在执行指令POP_JUMP_IF_FALSE时，会进行下面的操作：

```
// POP_JUMP_IF_FALSE

1、x = POP()

2、if not x : jump
```

POP_JUMP_IF_FALSE指令，其实就是将栈顶的元素（一般是刚进行逻辑运算的结果）Pop出来，然后判断其是否为False，如果是False，那么就跳转，否则什么事也不做。

3. List comprehension的语法

在刚看到List Comprehension时，很不能理解这个语法，总会有一个疑问：在List Comprehesion中，是否只能写for和if语句？能否写while语句？能否写try-except-finally语句？而且for语句和if语句之间的关系是什么？

有很多疑问，最终还得看Grammar/Grammer这个文件中，定义的语法规则。

其中，List comprehension的规则，在Grammar文件中，称为listmaker。

listmaker分为2种，最简单的一种，如下：

```
a = [1, 2, 3, 4, 5]
```

也就是直接列出List中的所有元素。这种方式最简单，也最好理解。

第二种就是本文所说的List Comprehension了，语法如下：

```
1、listmaker: test list_for

2、list_for: for’ explist ‘in’ testlist_safe [list_iter]

3、list_iter: list_for | list_if

4、list_if: if’ old_test [list_iter]
```

语法文件全是正则表达式，而且前后相互引用，读起来非常吃力。不过在上面所列的这4行语法规则中，可以看到：list comprehension中，只能使用for和if这2种语句。这也解决了一大部分疑问。

而且可以从上面的语法中看出，每个for语句后面，还可以接一个for语句或者一个if语句；每个if语句后面，也可以接一个for语句或者一个if语句；并且没有对for语句、if语句的个数有任何限制。

如果注意看上面关于list_for语法的规则，可以发现里面有一个叫testlist_safe的东西，这里要和Dictionary的推导语法规则对照一下，会发现很有趣的现象。

Dictionary推导式的一部分语法规则如下：

```
comp_for: 'for' exprlist 'in' or_test [ comp_for ]
```

这里的comp_for语法规则，几乎和上面的list_for语法规则相同，唯一不同的是在list_for语法规则中的testlist_safe位置上，变成了or_test。

只从字面看来，testlist_safe和or_test相比，中间有一个’ list’ 单词，也就是说：testlist_safe可以是一个列表，而or_test不可以，举例如下：

```
a = [ x for x in 1, 2, 3, 4 ]
```

在构造列表a时，可以直接在for … in …中，列出所有的元素，即上面的“1, 2, 3, 4”；

但是，如果是在构造一个Dictionary（或Set），如下：

```
a = { x for x in 1, 2, 3, 4 }
```

语法解析是会报错的！因为在Dictionary（或Set）的推导式语法规则中，for…in…中，不能是所有元素的列表！

为什么会有这种不同的语法呢？我还没搞明白！

另外，还会发现testlist_old，里面还一个”old”单词，这个很容易引起头疼，因为加了”old”这个单词，很有可能是为了和老版本兼容而出现的语法规则。而历史遗留问题，是最让人头疼的问题了。

在Python的语法规则里面，还有一个叫做testlist的语法规则，那么这个testlist_old中的”old”到底是什么意思呢？

经过几番考究，原来如下，一个简单的例子：

```
1 a = 5 if b > 3 else 2
```

上面是Python中类似C的三元运算符”?:”的语法，在Python内部，称为”if-expr”。

这个语法，就是没有”old”的语法。而在testlist_old语法中，这种带有if-else的语法是不允许的。

为什么呢？

例如，在list comprehension中，可以写成这样：

```
1 a = [ x for x in 5 if b > 3 else 2, 3 ]
```

在这个例子中，if-else本意上是：5 if b > 3 else 2，组成一个上面所说的”if-expr”语句的，但是却和List Comprehension中的for、if语法发生了冲突：这里这个if到底是List Comprehension中的呢？还是if-expr中的呢？

为了杜绝这种歧义，Python在语法规则中，就使用testlist_old而不是testlist，使得if-expr语句不能出现在for…in…的元素列表中。

4. List Comprehension中，for语句和if语句是什么关系呢？

Python的List Comprehension中，可以使用无限多个for、if语句，该怎么去理解这些for、if语句呢？它们之间的关系是什么呢？

Python的语法解析、字节码生成，大约分为3个阶段：

1、将.py源代码，解析成语法树

2、将语法树，解析成AST树

3、根据AST树，生成字节码

对于List Comprehension，可以在第2阶段，即Python/Ast.c这个源文件中，发现for语句和if语句之间的关系。

Python在从语法树生成AST的过程中，会将List Comprehension中的for分离开来，并将每个if语句，全部归属于离他最近的左边的for语句，例如：

```
a = [ (x, y) for x in range(10) if x % 2 if x > 3 for y in range(10) if y > 7 if
```

上面这段代码中，有2个for和4个if，分别如下：

1、for x in range(10)

2、for y in range(10)

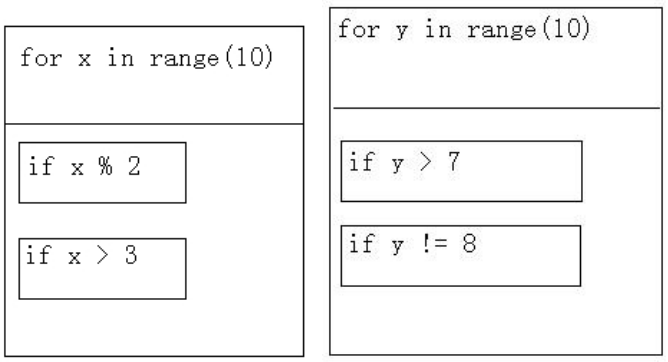
3、if x % 2

4、if x > 3

5、if y > 7

6、if y != 8

在AST的过程中，Python会按照for语句将上面的语句拆成2部分，分别如下：



每个if语句，从属于离他最近的左边的for语句。

下面看语法解析的第三阶段，即：通过AST生成字节码，在源代码Python/Compiler.c文件中。

在Python/Compiler.c源文件中，处理List Comprehension的代码，主要是2592行的compiler_listcomp_generator(…)函数。

这个函数，首先会生成字节码：BUILD_LIST，即生成一个新的List；然后通过自身的递归，从左到右，依次处理AST过程生成的for语句及其从属的if语句。

其中对于每一个for语句，大抵生成以下字节码：

```
1:      GET_ITER

2:      FOR_ITER
```

然后从左到右，依次处理从属与这个for语句的if语句，大抵生成以下字节码：

```
1: 进行逻辑判断并将结果Push进函数栈
2: POP_JUMP_IF_FALSE(如果结果为False，则跳转到XX处)

XX: 跳转到for语句的FOR_ITER处执行，相当于continue语句
```

也就是说，进入每一个for语句后，先取出Object的迭代器（通过GET_ITER），然后不断对其执行FOR_ITER指令，每次迭代出一个元素，都要对从属的if语句进行判断，如果有一个为False，相当于continue，返回FOR_ITER处，迭代下一个元素。

如果所有的if都判断为True，才进入后续的for语句中执行（后续的for语句都嵌套在之前的for语句中），直到最后一个for语句执行结束（从属的if都判断为True），这时才向list中append一个新的元素。

整个过程的伪码可以如下：

```
for xx in xxx:
    if not xxxx:
        continue;
    if not xxxxx:
        continue;

    ........

    for xx in xxx:
        if not xxxx:
            continue;

        ........

        // 到了最后一个for
        for xx in xxx:
            .......

            List.append(value)
```

至此，List Comprehension的内部运行过程就搞明白了。

5. 最后一个问题，列表推导式中的变量，是局部变量吗？

例如，一个简单的例子：

```
1 a = [ x for x in range(10) ]
```

这里面的变量x，是局部变量吗？在列表推导式结束后，还可以访问变量x吗？

Python的变量作用域，是在源代码Python/Symtable.c中实现的。

关于Python的变量作用域，打算再写一篇另外的文章介绍。

这里的结果是：如果变量x没有被使用过，那么变量x会成为一个局部变量，当列表推导式结束后，还可以访问变量x；否则，变量x原来的作用域是什么，现在还是什么。

(完)