2.15 字符串中插入变量¶

问题¶

你想创建一个内嵌变量的字符串，变量被它的值所表示的字符串替换掉。

解决方案¶

Python并没有对在字符串中简单替换变量值提供直接的支持。 但是通过使用字符串的 `format()` 方法来解决这个问题。比如：

```
>>> s = '{name} has {n} messages.'
>>> s.format(name='Guido', n=37)
'Guido has 37 messages.'
>>>
```

或者，如果要被替换的变量能在变量域中找到， 那么你可以结合使用 `format_map()` 和 `vars()` 。就像下面这样：

```
>>> name = 'Guido'
>>> n = 37
>>> s.format_map(vars())
'Guido has 37 messages.'
>>>
```

`vars()` 还有一个有意思的特性就是它也适用于对象实例。比如：

```
>>> class Info:
...     def __init__(self, name, n):
...         self.name = name
...         self.n = n
...
>>> a = Info('Guido',37)
>>> s.format_map(vars(a))
'Guido has 37 messages.'
>>>
```

`format` 和 `format_map()` 的一个缺陷就是它们并不能很好的处理变量缺失的情况，比如：

```
>>> s.format(name='Guido')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'n'
>>>
```

一种避免这种错误的方法是另外定义一个含有 `__missing__()` 方法的字典对象，就像下面这样：

```
class safesub(dict):
"""防止key找不到"""
def __missing__(self, key):
    return '{' + key + '}'
```

现在你可以利用这个类包装输入后传递给 `format_map()` ：

```
>>> del n # Make sure n is undefined
>>> s.format_map(safesub(vars()))
'Guido has {n} messages.'
>>>
```

如果你发现自己在代码中频繁的执行这些步骤，你可以将变量替换步骤用一个工具函数封装起来。就像下面这样：

```
import sys

def sub(text):
    return text.format_map(safesub(sys._getframe(1).f_locals))
```

现在你可以像下面这样写了：

```
>>> name = 'Guido'
>>> n = 37
>>> print(sub('Hello {name}'))
Hello Guido
>>> print(sub('You have {n} messages.'))
You have 37 messages.
>>> print(sub('Your favorite color is {color}'))
Your favorite color is {color}
>>>
```

讨论¶

多年以来由于Python缺乏对变量替换的内置支持而导致了各种不同的解决方案。 作为本节中展示的一个可能的解决方案，你可以有时候会看到像下面这样的字符串格式化代码：

```
>>> name = 'Guido'
>>> n = 37
>>> '%(name) has %(n) messages.' % vars()
'Guido has 37 messages.'
>>>
```

你可能还会看到字符串模板的使用：

```
>>> import string
>>> s = string.Template('$name has $n messages.')
>>> s.substitute(vars())
'Guido has 37 messages.'
>>>
```

然而， `format()` 和 `format_map()` 相比较上面这些方案而已更加先进，因此应该被优先选择。 使用 `format()` 方法还有一个好处就是你可以获得对字符串格式化的所有支持(对齐，填充，数字格式化等待)， 而这些特性是使用像模板字符串之类的方案不可能获得的。

本机还部分介绍了一些高级特性。映射或者字典类中鲜为人知的 `__missing__()` 方法可以让你定义如何处理缺失的值。 在 `SafeSub` 类中，这个方法被定义为对缺失的值返回一个占位符。 你可以发现缺失的值会出现在结果字符串中(在调试的时候可能很有用)，而不是产生一个 `KeyError` 异常。

`sub()` 函数使用 `sys._getframe(1)` 返回调用者的栈帧。可以从中访问属性 `f_locals` 来获得局部变量。 毫无疑问绝大部分情况下在代码中去直接操作栈帧应该是不推荐的。 但是，对于像字符串替换工具函数而言它是非常有用的。 另外，值得注意的是 `f_locals` 是一个复制调用函数的本地变量的字典。 尽管你可以改变 `f_locals` 的内容，但是这个修改对于后面的变量访问没有任何影响。 所以，虽说访问一个栈帧看上去很邪恶，但是对它的任何操作不会覆盖和改变调用者本地变量的值。