使用__future__

1530次阅读

Python的每个新版本都会增加一些新的功能，或者对原来的功能作一些改动。有些改动是不兼容旧版本的，也就是在当前版本运行正常的代码，到下一个版本运行就可能不正常了。

从Python 2.7到Python 3.x就有不兼容的一些改动，比如2.x里的字符串用'xxx'表示str，Unicode字符串用u'xxx'表示unicode，而在3.x中，所有字符串都被视为unicode，因此，写u'xxx'和'xxx'是完全一致的，而在2.x中以'xxx'表示的str就必须写成b'xxx'，以此表示“二进制字符串”。

要直接把代码升级到3.x是比较冒进的，因为有大量的改动需要测试。相反，可以在2.7版本中先在一部分代码中测试一些3.x的特性，如果没有问题，再移植到3.x不迟。

Python提供了__future__模块，把下一个新版本的特性导入到当前版本，于是我们就可以在当前版本中测试一些新版本的特性。举例说明如下：

为了适应Python 3.x的新的字符串的表示方法，在2.7版本的代码中，可以通过unicode_literals来使用Python 3.x的新的语法：

```
# still running on Python 2.7

from __future__ import unicode_literals

print '\'xxx\' is unicode?', isinstance('xxx', unicode)
print 'u\'xxx\' is unicode?', isinstance(u'xxx', unicode)
print '\'xxx\' is str?', isinstance('xxx', str)
print 'b\'xxx\' is str?', isinstance(b'xxx', str)
```

注意到上面的代码仍然在Python 2.7下运行，但结果显示去掉前缀u的'a string'仍是一个unicode，而加上前缀b的b'a string'才变成了str：

```
$ python task.py
'xxx' is unicode? True
u'xxx' is unicode? True
'xxx' is str? False
b'xxx' is str? True
```

类似的情况还有除法运算。在Python 2.x中，对于除法有两种情况，如果是整数相除，结果仍是整数，余数会被扔掉，这种除法叫“地板除”：

```
>>> 10 / 3
3
```

要做精确除法，必须把其中一个数变成浮点数：

```
>>> 10.0 / 3
3.3333333333333335
```

而在Python 3.x中，所有的除法都是精确除法，地板除用//表示：

```
$ python3
Python 3.3.2 (default, Jan 22 2014, 09:54:40)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 / 3
3.3333333333333335
```

```
>>> 10 // 3
3
```

如果你想在Python 2.7的代码中直接使用Python 3.x的除法，可以通过__future__模块的division实现：

```
from __future__ import division

print '10 / 3 =', 10 / 3
print '10.0 / 3 =', 10.0 / 3
print '10 // 3 =', 10 // 3
```

结果如下：

```
10 / 3 = 3.33333333333
10.0 / 3 = 3.33333333333
10 // 3 = 3
```

小结

由于Python是由社区推动的开源并且免费的开发语言，不受商业公司控制，因此，Python的改进往往比较激进，不兼容的情况时有发生。Python为了确保你能顺利过渡到新版本，特别提供了__future__模块，让你在旧的版本中试验新版本的一些特性。
