漫谈Python语言

Hoxide

March 8, 2005

内容提要 I

什么是Python

Note:

Python语言可能是第一种即简单又功能强大的编程语言。它不仅适合于初学者，也适合于专业人员使用，更加重要的是，用Python编程是一种愉快的事。本身将帮助你学习这个奇妙的语言，并且向你展示如何即快捷又方便地完成任务——真正意义上“为编程问题提供的完美解决方案！”

– 《简明Python 教程》Swaroop, C. H. 著 沈洁元译

Python的特点

- Python 是荷兰人Guido van Rossum 1989年写的一个脚本语言。
- Python 是基于字节码的动态解释语言
- Python 支持OOP
- Python 是可以扩展和嵌入的
- 支持函数式程序设计
- Python 是开放的Python 本身源代码开放很多python 模块提供源代码
- 平台无关的多线程支持

Python的影响力

Note:

limodou

joit卓越奖提名中有python:

- *Borland Delphi 2005 (Borland)*
- *CodeRush 1.1 for Visual Studio (Developer Express)*
- *Eclipse 3.0 (Eclipse Foundation)*
- *IntelliJ IDEA4.5 (JetBrains)*
- *JBoss AOP 1.0 (JBoss)*
- *Python2.4 (Python.org)*
- *REALbasic 5.5 for Windows Professional Edition (REAL Software)*
- *Sun Java Studio Enterprise 7 (Sun Microsystems)*

Python的影响力

Note:

limodou评论: 这是从CSDN上拷贝了一段，注意是提名，最终结果还不知道。不过相比之外python有些简陋，上面有好几个是集成环境。好在奖项的名字是开发语言和开发环境。不过这样说来放在一起也有些不合适啊。

The Zen of Python I

The Zen of Python
蟒禅
Beautiful is better than ugly.
美丽好过丑陋;
Explicit is better than implicit.
明显好过隐晦;
Simple is better than complex.
简单好过复合;
Complex is better than complicated.
复合好过复杂;
Flat is better than nested. 扁平好过嵌套;
Sparse is better than dense.
稀疏好过密集;
Readability counts.
可读性最重要;

The Zen of Python II

Special cases aren't special enough to break the rules.
即便实用性比纯度重要,
Although practicality beats purity.
但是!特殊案例不可特殊到打破规则;
Errors should never pass silently.
错误从来不会默默消失,
Unless explicitly silenced.
直到明确的让它闭嘴!
In the face of ambiguity,
面对模糊,
refuse the temptation to guess.
拒绝猜测的诱惑;
There should be one– and preferably only one –obvious way to do it.
应该有一个(宁愿只有一个)显而易见的解决方法;
Although that way may not be obvious at first unless you're Dutch.

The Zen of Python III

尽管刚开始方法不会是很明显,除非你是(Dutch)

Now is better than never. Although never is often better than *right* now.

即使永远不做比"立刻"做要"聪明", 但是！现在就做永远比不做要好;

If the implementation is hard to explain, it's a bad idea.

只要实现很难解释,那么它就不是一个好主意;

If the implementation is easy to explain, it may be a good idea.

只要实现很容易解释,那么这就是一个好主意;

Namespaces are one honking great idea

名称空间是一个正在召唤的绝妙想法

– let's do more of those!

–大家一起来实践这些规则吧!

– by Tim Peters

Code:

```
import os
from os.path import join, getsize
import sys

print sys.argv[1]
for root, dirs, files in os.walk (sys.argv[1]):
    if 'CVS' in dirs:
        fn = join (root + '\CVS', 'ROOT')
        print root + ' :', fn
        f = open (fn, 'r')
        r = f.read ()
        if r.startswith ('e:\cvsroot'):
            open (fn, 'w').write ('g:\cvsroot')
            f = open (fn, 'r')
            r = f.read ()
            print r
```

Code:

```
import os
from os.path import join, getsize
import sys

print sys.argv[1]
for root, dirs, files ∈ os.walk (sys.argv[1]):
    if 'CVS' ∈ dirs:
        fn ← join (root + '\CVS', 'ROOT')
        print root + '␣:', fn
        f ← open (fn, 'r')
        r ← f.read ()
        if r.startswith ('e:\cvsroot'):
            open (fn, 'w').write ('g:\cvsroot')
            f ← open (fn, 'r')
            r ← f.read ()
            print r
```

Python的发行版本

虽然通常所说的Python是指Guido van Rossum 用C语言实现的CPython, 但是其实还有一些其他的很有前途的Python实现. 大致分有以下一些版本:

- CPython — Guido van Rossum
- Jython — Jim Hugunin
- IronPython — Jim Hugunin
- Python.net — Jim Hugunin — MS

Note:

下面的代码都在CPython上运行通过

CPython

CPython也就是通常说的Python, 他最先由Guido van Rossum在90年代早期开发的. 目前Guido van Rossum仍然是主要的开发者. CPython的当前版本是2.4, 但是很多库仍然没有windows上的2.4发布版本, 因此如果在windows上我推荐2.3.4.

目前CPython在语法方面的改变已经非常少, 属于非常成熟的语言了. 它可以在各种Unix, Windows, Mac上运行

可从`http://www.python.org/`得到更多信息

Jython

Jython是Python的Java实现，可以让用户把Python源代码编译为Java的字节代码（byte code），而在任何Java虚拟机上运行。它能和Java无缝集成，通过Python可以完全存取所有的Java库，建立applets，并且和Java beans集成，并把Java类细分（subclass）。和Python一样，Jython可以交互使用，但是Java没有这个功能。

可以从Jython网站得到更多资料：`http://www.jython.org/`

IronPython

Note:

号称比CPython还快

IronPython的作者为Jim Hugunin，他同时也是Jython的作者，Jython是一个在Java平台上Python的实现。曾有开源社区的开发者认为.NET不是一个很好的动态语言实现平台。但是事实证明.NET可以很有效率的运行动态语言所生成的代码。IronPython这还仅仅是在.NET 1.1上就有很不错的成绩，据悉.NET 2.0将会强化对动态语言的支持，我们有理由期待Python在.NET平台上更好的表现。

Python.net

由Microsoft主导开发的.NET环境下的Python语言实现, 仍然由Jim Hugunin主持. 可以方便得访问CLR, 市IronPython的后继者. 目前版本1.0 – *beta*4, 他其实是Python2.4的一组扩展库, 简单的在Python2.4的安装目录中加入这些库就可以方便得使用Python.net.

Note:

Python2.4的windows版本由MSVC7.0编译.

Note:

例子

类型系统

- 基于对象模型, 一切皆对象, 强类型
- 包和库, 名字空间
- 多重继承, Mix-In(混入), 自省
- 丰富的基本类型, 包括数字, 序列, 映射, 文件和Python内部的一些类型
- 基于虚拟机技术, python字节码
- 垃圾收集
- 与$c/c++$的完美配合

基本类型— 数字

- 整型
  - Plain integers, -2147483648 到2147483647
  - Long integers, 无位数限制
  - Booleans, 逻辑型True False
- 浮点
- 复数

复合类型

- 简单序列
  - String
  - Unicode
  - Tuples (元组)
- 复合序列— List (列表)
- Mapping(映射) — Dictionary (字典)

例子 I

Note:

如何运行这些例子, 你可以简单的在命令行中输入'python' 来打开Python解释器. 如果是还有图形界面的标准Python IDE — IDLE, windows安装用户可以从开始→程序→Python→IDLE 打开.

例子 II

Code:

```
>>> t = (1, 2, 3)
>>> type(t)
< type 'tuple' >
>>> t[1]
2
>>> a = "hello"
>>> a
"hello"
>>> a[-1]
'o'
```

例子 III

Code:

```
>>> b = list(a)
>>> b
['h', 'e', 'l', 'l', 'o']
>>> b[4:]
['o']
>>> b[3:-1]
['l']
>>> b[1:2] = ['a','b','c']
>>> b
>>> ['h','a','b','c','l','l','o']
```

例子 IV

Code:

```
>>> d = dict(zip(b, range(len(b))))
>>> d
{'h': 0, 'e': 1, 'l': 3, 'o': 4}
>>> d.keys()
['h', 'e', 'l', 'o']
>>> d.items()
[('h', 0), ('e', 1), ('l', 3), ('o', 4)]
```

类 I

- class
- 多重继承
- Mix-in(混入)

类 II

Code:

```
>>> class c1 :
        def __init__(self) :
            self.message = 'init'
>>> i1 = c1()
>>> i1.message
'init'
```

Code:

```
>>> def __init__(self, message) :
        self.message = message
>>> c1.__init__ = __init__
>>> l2 = c1('hello')
>>> l2.message
'hello'
```

类 III

Code:

```
>>> class c2(c1):
        def __del__(self):
            print 'del c2 instance', self.message
>>> def a():
        ls = c2('hello')
>>> a()
del c2 instance hello
```

类 IV

Code:

```
>>> class c3 :
        def __call__(self) :
            print self.message
>>> l4 = c1('hello')
>>> c1.__bases__ += (c3,)
>>> l4()
hello
```

类 V

Code:

```
>>> class c4 :
        def __call__(self) :
            self.__class__ = c5
            print 'c4'
>>> class c5 :
        def __call__(self) :
            self.__class__ = c4
            print 'c5'
>>> l4 = c4()
>>> l4()
c4
>>> l4()
c5
```

关于类的魔法元素 I

Python中一般以“__”包围的变量都有特殊含义.

- *__doc__*
- *__class__*
- *__bases__*
- *__module__*

Note:

关于“__doc__”即Document String, 可以用来生成文档, 工具有python标准模块pydoc和非常流行的epydoc.

其他魔法元素 I

- *__main__*
- *__builtins__*

Code:

```
if __name__ == '__main__':
    print __name__
```

Code:

```
>>> __builtins__
<module '__builtin__' (built-in)>
```

还有很多带有“魔法”的元素, 可以参见Python Manual

基本结构

- while
- for
- break

例子 I

Code:

```
>>> l = range(10)
>>> while l :
        print l.pop(),
9 8 7 6 5 4 3 2 1 0
>>> for i in range(9, -1, -1) :
        print i,
9 8 7 6 5 4 3 2 1 0
```

生成器

Python支持生成器, 你可以把他简单得看成一个模拟iter(叠代器)的函数, 使用'yield'关键字.

Code:

```
>>> def g():
        l = range(10)
        while l:
            yield l.pop()
>>> l = g()
>>> for i in l:
        print i,
9 8 7 6 5 4 3 2 1 0
```

异常 I

- try
- except
- finally
- raise

Code:

```
>>> l = g()
>>> while True :
        try :
            print l.next(),
        except StopIteration :
            break
9 8 7 6 5 4 3 2 1 0
```

匿名函数 I

- lambda
- map
- reduce
- filter
- eval

匿名函数 II

Python部分支持函数编程的功能, 提供关键字'lambda'创建匿名函数. 结合map, reduce, filter和谓词(and, or) 等可以创建及其复杂的函数式的程序. 你可以混用一般的过程式风格和函数式风格, 选择更简洁的方式表达算法.

Code:

```
>>> map(str, range(10))
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
>>> from random import random as _random
>>> [i for i in range(10) if i > 5]
[6, 7, 8, 9]
>>> filter(lambda x: x > 0.5 and x, [_random() for i in range(10)])
[0.9938822073011242, 0.9420340085030887,
0.9951802578009955, 0.7993882930938341]
```

匿名函数 III

Code:

```
>>> def e():
        r = raw_input('pleaseinput : ')
        print eval(r)
    e()
pleaseinput : 1 + 2
3
```

Note:

http://wiki.woodpecker.org.cn/moin.cgi/PyPorgramGames 上的游戏收集中有hoxide写的解24点问题和爱因司坦难题的程序, 使用了很多函数式的方法, 是不错的例子.

了解类型 I

- *type* — 获取类型
- *dir* — 列出对象所含的属性和方法
- *traceback* — 访问调用栈

了解类型 II

Code:

```
>>> dir()
['__builtins__', '__doc__', '__name__']
```

了解类型 III

Code:

```
>>> from traceback import extract_stack
>>> def a() :
        b()
>>> def b() :
        fname = extract_stack()[-2][2]
        print 'caller :', fname
        print eval(fname)
>>> a()
caller : a
< function at 0x00A56370 >
```