目录

0.1. 关于本书

"Those people who have nothing better to do than post on the Internet all day long are rarely the ones who have the most insights."
- Jakob Nielsen, December 1998

五年前我偶然遇到了 Python, 开始了我的 Python 之旅, 我花费了大量的时间在 comp.lang.python 新闻组里回答问题. 也许某个人发现一个模块正是他想要的, 但是却不知道如何使用它. 也许某个人为他的任务挑选的不合适的模块. 也许某个人已经厌倦了发明新轮子. 大多时候, 一个简短的例子要比一份手册文档更有帮助.

本书是超过 3,000 个新闻组讨论的精华部分, 当然也有很多的新脚本, 为了涵盖标准库的每个角落.

我尽力使得每个脚本都易于理解, 易于重用代码. 我有意缩短注释的长度, 如果你想更深入地了解背景, 那么你可以参阅每个 Python 发布中的参考手册. 本书的重要之处在于范例代码.

我们欢迎任何评论, 建议, 以及 bug 报告, 请将它们发送到 fredrik@pythonware.com. 我将阅读尽我所能阅读所有的邮件, 但可能回复不是那么及时.

本书的相关更新内容以及其他信息请访问 http://www.pythonware.com/people/fredrik/librarybook.htm

为什么没有 Tkinter?

本书涵盖了整个标准库, 除了(可选的)Tkinter ui(user-interface : 用户界面)库. 有很多原因, 更多是因为时间, 本书的空间, 以及我正在写另一本关于 Tkinter 的书.

关于这些书的信息, 请访问 http://www.pythonware.com/people/fredrik/tkinterbook.htm. (不用看了, 又一 404)

产品细节

本书使用 DocBook SGML 编写, 我使用了一系列的工具, 包括 Secret Labs' PythonWorks, Excosoft Documentor, James Clark's Jade DSSSL processor, Norm Walsh's DocBook stylesheets, 当然,还有一些 Python 脚本.

感谢帮忙校对的人们: Tim Peters, Guido van Rossum, David Ascher, Mark Lutz, 和 Rael Dornfest, 以及 PythonWare 成员: Matthew Ellis, Håkan Karlsson, 和 Rune Uhlin.

感谢 Lenny Muellner, 他帮助我把 SGML 文件转变为你们现在所看到的这本书, 以及 Christien Shangraw, 他将那些代码文件集合起来做成了随书 CD (可以在 http://examples.oreilly.com/pythonsl 找到, 竟然没有 404, 奇迹).

0.2. 代码约定

本书使用以下习惯用法:

斜体

用于文件名和命令. 还用于定义术语.

等宽字体 `e.g. Python`

用于代码以及方法,模块,操作符,函数,语句,属性等的名称.

等宽粗体

用于代码执行结果.

0.3. 关于例子

除非提到,所有例子都可以在 Python 1.5.2 和 Python 2.0 下运行. 能不能在 Python 2.4/2.5 下执行.....看参与翻译各位的了.

除了一些平台相关模块的脚本, 所有例子都可以在 Windows, Solaris, 以及 Linux 下正常执行.

所有代码都是有版权的. 当然,你可以自由地使用这些这些模块,别忘记你是从哪得到(?学会)这些的.

大多例子的文件名都包含它所使用的模块名称,后边是 `"-example-"` 以及一个唯一的"序号". 注意有些例子并不是按顺序出现的, 这是为了匹配本书的较早版本 - *(the eff-bot guide to) The Standard Python Library*.

你可以在网上找到本书附带 CD 的内容 (参阅 http://examples.oreilly.com/pythonsl). 更多信息以及更新内容参阅 http://www.pythonware.com/people/fredrik/librarybook.htm. (ft, 又一 404. 大家一定不要看~)

0.4. 如何联系我们

Python 江湖 QQ 群: 43680167

Feather (校对) QQ: 85660100

---

1. 核心模块

"Since the functions in the C runtime library are not part of the Win32 API, we believe the number of applications that will be affected by this bug to be very limited."
- Microsoft, January 1999

1.1. 介绍

Python 的标准库包括了很多的模块, 从 Python 语言自身特定的类型和声明, 到一些只用于少数程序的不著名的模块.

本章描述了一些基本的标准库模块. 任何大型 Python 程序都有可能直接或间接地使用到这类模块的大部分.

1.1.1. 内建函数和异常

下面的这两个模块比其他模块加在一起还要重要: 定义内建函数(例如 len, int, range ...)的 `__builtin__` 模块, 以及定义所有内建异常的 `exceptions` 模块.

Python 在启动时导入这两个模块, 使任何程序都能够使用它们.

1.1.2. 操作系统接口模块

Python 有许多使用了 POSIX 标准 API 和标准 C 语言库的模块. 它们为底层操作系统提供了平台独立的接口.

这类的模块包括: 提供文件和进程处理功能的 `os` 模块; 提供平台独立的文件名处理 (分拆目录名, 文件名, 后缀等)的 `os.path` 模块; 以及时间日期处理相关的 `time/datetime` 模块.

> [!Feather 注: datetime 为 Py2.3 新增模块, 提供增强的时间处理方法]

延伸一点说, 网络和线程模块同样也可以归为这一个类型. 不过 Python 并没有在所有的平台/版本实现这些.

1.1.3. 类型支持模块

标准库里有许多用于支持内建类型操作的库. `string` 模块实现了常用的字符串处理. `math` 模块提供了数学计算操作和常量(pi, e 都属于这类常量), `cmath` 模块为复数提供了和 `math` 一样的功能.

1.1.4. 正则表达式

`re` 模块为 Python 提供了正则表达式支持. 正则表达式是用于匹配字符串或特定子字符串的有特定语法的字符串模式.

1.1.5. 语言支持模块

sys 模块可以让你访问解释器相关参数,比如模块搜索路径,解释器版本号等. `operator` 模块提供了和内建操作符作用相同的函数. `copy` 模块允许你复制对象, Python 2.0 新加入的 `gc` 模块提供了对垃圾收集的相关控制功能.

1.2. _ _builtin_ _ 模块

这个模块包含 Python 中使用的内建函数. 一般不用手动导入这个模块; Python 会帮你做好一切.

1.2.1. 使用元组或字典中的参数调用函数

Python 允许你实时地创建函数参数列表. 只要把所有的参数放入一个元组中, 然后通过内建的 `apply` 函数调用函数. 如 Example 1-1.

1.2.1.1. Example 1-1. 使用 apply 函数

> File: builtin-apply-example-1.py

```
def function(a, b):
    print a, b

apply(function, ("whither", "canada?"))
apply(function, (1, 2 + 3))
```

```
whither canada?
1 5
```

要想把关键字参数传递给一个函数，你可以将一个字典作为 apply 函数的第 3 个参数，参考 Example 1-2.

1.2.1.2. Example 1-2. 使用 apply 函数传递关键字参数

```
File: builtin-apply-example-2.py

def function(a, b):
    print a, b

apply(function, ("crunchy", "frog"))
```

```
apply(function, ("crunchy",), {"b": "frog"})

apply(function, (), {"a": "crunchy", "b": "frog"})
```

```
crunchy frog

crunchy frog

crunchy frog
```

apply 函数的一个常见用法是把构造函数参数从子类传递到基类, 尤其是构造函数需要接受很多参数的时候. 如 Example 1-3 所示.

1.2.1.3. Example 1-3. 使用 apply 函数调用基类的构造函数

```
File: builtin-apply-example-3.py
```

```
class Rectangle:
```

```
    def _ _init_ _(self, color="white", width=10,
height=10):

        print "create a", color, self, "sized",
width, "x", height


class RoundedRectangle(Rectangle):
```

```
    def _ _init_ _(self, **kw):

        apply(Rectangle._ _init_ _, (self,), kw)



rect = Rectangle(color="green", height=100,
width=100)

rect = RoundedRectangle(color="blue", height=20)



create a green <Rectangle instance at 8c8260> sized
100 x 100

create a blue <RoundedRectangle instance at 8c84c0>
sized 10 x 20
```

Python 2.0 提供了另个方法来做相同的事. 你只需要使用一个传统的函数调用 , 使用 * 来标记元组, ** 来标记字典.

下面两个语句是等价的:

```
result = function(*args, **kwargs)

result = apply(function, args, kwargs)
```

1.2.2. 加载和重载模块

如果你写过较庞大的 Python 程序, 那么你就应该知道 import 语句是用来导入外部模块的 (当然也可以使用 from-import 版本). 不过你可能不知道 import 其实是靠调用内建函数 _ _import_ _ 来工作的.

通过这个戏法你可以动态地调用函数. 当你只知道模块名称(字符串)的时候, 这将很方便. Example 1-4 展示了这种用法, 动态地导入所有以 "-plugin" 结尾的模块.

1.2.2.1. Example 1-4. 使用 _ _import_ _ 函数加载模块

```
File: builtin-import-example-1.py



import glob, os



modules = []



for module_file in glob.glob("*-plugin.py"):

    try:
```

```
        module_name, ext =
os.path.splitext(os.path.basename(module_file))
```

```
        module = _ _import_ _(module_name)

        modules.append(module)

    except ImportError:

        pass # ignore broken modules
```

```
# say hello to all modules

for module in modules:

    module.hello()



example-plugin says hello
```

注意这个 plug-in 模块文件名中有个 "-" (hyphens). 这意味着你不能使用普通的 import 命令, 因为 Python 的辨识符不允许有 "-" .

Example 1-5 展示了 Example 1-4 中使用的 plug-in .

1.2.2.2. Example 1-5. Plug-in 例子

```
File: example-plugin.py



def hello():

    print "example-plugin says hello"
```

Example 1-6 展示了如何根据给定模块名和函数名获得想要的函数对象.

1.2.2.3. Example 1-6. 使用 _ _import_ _ 函数获得特定函数

```
File: builtin-import-example-2.py


```

```
def getfunctionbyname(module_name, function_name):
    module = _ _import_ _(module_name)
    return getattr(module, function_name)


print repr(getfunctionbyname("dumbdbm", "open"))


<function open at 794fa0>
```

你也可以使用这个函数实现延迟化的模块导入 (lazy module loading). 例如在 Example 1-7 中的 `string` 模块只在第一次使用的时候导入.

1.2.2.4. Example 1-7. 使用 _ _import_ _ 函数实现 延迟导入

```
File: builtin-import-example-3.py
```

```

```

```
class LazyImport:
    def _ _init_ _(self, module_name):
        self.module_name = module_name
        self.module = None
    def _ _getattr_ _(self, name):
```

```
        if self.module is None:

            self.module = _ _import_
_(self.module_name)

        return getattr(self.module, name)



string = LazyImport("string")



print string.lowercase



abcdefghijklmnopqrstuvwxyz
```

Python 也提供了重新加载已加载模块的基本支持. [Example 1-8 #eg-1-8 会加载 3 次 *hello.py* 文件.

1.2.2.5. Example 1-8. 使用 reload 函数

```
File: builtin-reload-example-1.py



import hello

reload(hello)

reload(hello)
```

```
hello again, and welcome to the show

hello again, and welcome to the show

hello again, and welcome to the show
```

reload 直接接受模块作为参数.

```
[!Feather 注:  ^ 原句无法理解, 稍后讨论.]
```

注意,当你重加载模块时, 它会被重新编译, 新的模块会代替模块字典里的老模块. 但是, 已经用原模块里的类建立的实例仍然使用的是老模块(不会被更新). 同样地, 使用 `from-import` 直接创建的到模块内容的引用也是不会被更新的.

1.2.3. 关于名称空间

`dir` 返回由给定模块, 类, 实例, 或其他类型的所有成员组成的列表. 这可能在交互式 Python 解释器下很有用, 也可以用在其他地方. Example 1-9 展示了 `dir` 函数的用法.

1.2.3.1. Example 1-9. 使用 dir 函数

```
File: builtin-dir-example-1.py



def dump(value):

    print value, "=>", dir(value)
```

```
import sys



dump(0)

dump(1.0)

dump(0.0j) # complex number

dump([]) # list

dump({}) # dictionary

dump("string")

dump(len) # function
```

```
dump(sys) # module
```

```
0 => []

1.0 => []

0j => ['conjugate', 'imag', 'real']

[] => ['append', 'count', 'extend', 'index',
'insert',
```

```
    'pop', 'remove', 'reverse', 'sort']

{} => ['clear', 'copy', 'get', 'has_key', 'items',

    'keys', 'update', 'values']

string => []

<built-in function len> => ['__doc__', '__name__',
'__self__']

<module 'sys' (built-in)> => ['__doc__', '__name__',

    '__stderr__', '__stdin__', '__stdout__',
'argv',

    'builtin_module_names', 'copyright',
'dllhandle',

    'exc_info', 'exc_type', 'exec_prefix',
'executable',

...
```

在例子 Example 1-10 中定义的 `getmember` 函数返回给定类定义的所有类级别的属性和方法.

1.2.3.2. Example 1-10. 使用 dir 函数查找类的所有成员

```
File: builtin-dir-example-2.py
```

```
class A:

    def a(self):

        pass

    def b(self):

        pass



class B(A):

    def c(self):

        pass

    def d(self):

        pass
```

```

```

```
def getmembers(klass, members=None):

    # get a list of all class members, ordered by
class

    if members is None:

        members = []
```

```
    for k in klass._ _bases_ _:
        getmembers(k, members)
    for m in dir(klass):
        if m not in members:
            members.append(m)
    return members

print getmembers(A)
print getmembers(B)
print getmembers(IOError)
```

```
['_ _doc_ _', '_ _module_ _', 'a', 'b']
```

```
['_ _doc_ _', '_ _module_ _', 'a', 'b', 'c', 'd']
```

```
['_ _doc_ _', '_ _getitem_ _', '_ _init_ _', '_
_module_ _', '_ _str_ _']
```

getmembers 函数返回了一个有序列表. 成员在列表中名称出现的越早, 它所处的类层次就越高. 如果无所谓顺序的话, 你可以使用字典代替列表.

[!Feather 注: 字典是无序的, 而列表和元组是有序的, 网上有关于有序字典的讨论]

vars 函数与此相似, 它返回的是包含每个成员当前值的字典. 如果你使用不带参数的 vars , 它将返回当前局部名称空间的可见元素(同 locals() 函数). 如 Example 1-11 所表示.

1.2.3.3. Example 1-11. 使用 vars 函数

File: builtin-vars-example-1.py

```
book = "library2"

pages = 250

scripts = 350
```

```
print "the %(book)s book contains more
than %(scripts)s scripts" % vars()
```

```
the library book contains more than 350 scripts
```

1.2.4. 检查对象类型

Python 是一种动态类型语言，这意味着给一个定变量名可以在不同的场合绑定到不同的类型上. 在接下面例子中，同样的函数分别被整数，浮点数，以及一个字符串调用:

```
def function(value):

    print value

function(1)

function(1.0)

function("one")
```

type 函数（如 Example 1-12 所示）允许你检查一个变量的类型. 这个函数会返回一个 *type descriptor (类型描述符)*，它对于 Python 解释器提供的每个类型都是不同的.

1.2.4.1. Example 1-12. 使用 type 函数

```
File: builtin-type-example-1.py



def dump(value):

    print type(value), value



dump(1)

dump(1.0)
```

```
dump("one")



<type 'int'> 1

<type 'float'> 1.0

<type 'string'> one
```

每个类型都有一个对应的类型对象，所以你可以使用 `is` 操作符（对象身份?）来检查类型.（如 Example 1-13 所示）.

1.2.4.2. Example 1-13. 对文件名和文件对象使用 type 函数

```
File: builtin-type-example-2.py



def load(file):

    if isinstance(file, type("")):

        file = open(file, "rb")

    return file.read()



print len(load("samples/sample.jpg")), "bytes"

print len(load(open("samples/sample.jpg", "rb"))),
"bytes"
```

```
4672 bytes

4672 bytes
```

callable 函数，如 Example 1-14 所示，可以检查一个对象是否是可调用的（无论是直接调用或是通过 apply）. 对于函数，方法，lambda 函式，类，以及实现了 __call__ 方法的类实例，它都返回 True.

1.2.4.3. Example 1-14. 使用 callable 函数

```
File: builtin-callable-example-1.py


def dump(function):
    if callable(function):
        print function, "is callable"
    else:
        print function, "is *not* callable"


class A:
    def method(self, value):
```

```
        return value



class B(A):

    def _ _call_ _(self, value):

        return value



a = A()

b = B()



dump(0) # simple objects

dump("string")

dump(callable)

dump(dump) # function



dump(A) # classes

dump(B)

dump(B.method)
```

```
dump(a) # instances
dump(b)
dump(b.method)
```

```
0 is *not* callable
string is *not* callable
<built-in function callable> is callable
<function dump at 8ca320> is callable
A is callable
B is callable
```

```
<unbound method A.method> is callable
```

```
<A instance at 8caa10> is *not* callable
<B instance at 8cab00> is callable
<method A.method of B instance at 8cab00> is callable
```

注意类对象 (A 和 B) 都是可调用的; 如果调用它们, 就产生新的对象(类实例). 但是 A 类的实例不可调用, 因为它的类没有实现 _ _call_ _ 方法.

你可以在 operator 模块中找到检查对象是否为某一内建类型(数字, 序列, 或者字典等) 的函数. 但是, 因为创建一个类很简单(比如实现基本序列方法的类), 所以对这些类型使用显式的类型判断并不是好主意.

在处理类和实例的时候会复杂些. Python 不会把类作为本质上的类型对待; 相反地, 所有的类都属于一个特殊的类类型(special class type), 所有的类实例属于一个特殊的实例类型(special instance type).

这意味着你不能使用 type 函数来测试一个实例是否属于一个给定的类; 所有的实例都是同样的类型! 为了解决这个问题, 你可以使用 isinstance 函数,它会检查一个对象是不是给定类(或其子类)的实例. Example 1-15 展示了 isinstance 函数的使用.

1.2.4.4. Example 1-15. 使用 isinstance 函数

```
File: builtin-isinstance-example-1.py



class A:

    pass
```

```

```

```
class B:

    pass



class C(A):

    pass
```

```
class D(A, B):
    pass


def dump(object):
    print object, "=>",
    if isinstance(object, A):
        print "A",
    if isinstance(object, B):
        print "B",
    if isinstance(object, C):
        print "C",
    if isinstance(object, D):
        print "D",
    print


a = A()
b = B()
```

```
c = C()

d = D()



dump(a)

dump(b)

dump(c)

dump(d)

dump(0)

dump("string")
```

```
<A instance at 8ca6d0> => A
```

```
<B instance at 8ca750> => B

<C instance at 8ca780> => A C

<D instance at 8ca7b0> => A B D

0 =>

string =>
```

issubclass 函数与此相似，它用于检查一个类对象是否与给定类相同，或者是给定类的子类. 如 Example 1-16 所示.

注意，isinstance 可以接受任何对象作为参数，而 issubclass 函数在接受非类对象参数时会引发 *TypeError* 异常.

1.2.4.5. Example 1-16. 使用 issubclass 函数

```
File: builtin-issubclass-example-1.py



class A:

    pass



class B:

    pass
```

```

```

```
class C(A):

    pass



class D(A, B):

    pass
```

```
def dump(object):
    print object, "=>",
    if issubclass(object, A):
        print "A",
    if issubclass(object, B):
        print "B",
    if issubclass(object, C):
        print "C",
    if issubclass(object, D):
        print "D",
    print
```

```
dump(A)
dump(B)
dump(C)
dump(D)
dump(O)
```

```
dump("string")



A => A

B => B

C => A C

D => A B D

0 =>

Traceback (innermost last):

  File "builtin-issubclass-example-1.py", line 29,
in ?

  File "builtin-issubclass-example-1.py", line 15,
in dump

TypeError: arguments must be classes
```

1.2.5. 计算 Python 表达式

Python 提供了在程序中与解释器交互的多种方法. 例如 `eval` 函数将一个字符串作为 Python 表达式求值. 你可以传递一串文本, 简单的表达式, 或者使用内建 Python 函数. 如 Example 1-17 所示.

1.2.5.1. Example 1-17. 使用 eval 函数

```
File: builtin-eval-example-1.py
```

```
def dump(expression):
    result = eval(expression)
    print expression, "=>", result, type(result)

dump("1")
dump("1.0")
dump("'string'")
dump("1.0 + 2.0")
dump("'*' * 10")
dump("len('world')")
```

```
1 => 1 <type 'int'>
1.0 => 1.0 <type 'float'>
'string' => string <type 'string'>
1.0 + 2.0 => 3.0 <type 'float'>
'*' * 10 => ********** <type 'string'>
```

```
len('world') => 5 <type 'int'>
```

如果你不确定字符串来源的安全性，那么你在使用 eval 的时候会遇到些麻烦. 例如，某个用户可能会使用 _ _import_ _ 函数加载 os 模块，然后从硬盘删除文件（如 Example 1-18 所示).

1.2.5.2. Example 1-18. 使用 eval 函数执行任意命令

```
File: builtin-eval-example-2.py



print eval("_ _import_ _('os').getcwd()")

print eval("_ _import_ _('os').remove('file')")



/home/fredrik/librarybook
```

```
Traceback (innermost last):
```

```
 File "builtin-eval-example-2", line 2, in ?

 File "<string>", line 0, in ?

os.error: (2, 'No such file or directory')
```

这里我们得到了一个 *os.error* 异常，这说明 *Python 事实上在尝试删除文件!*

幸运地是，这个问题很容易解决. 你可以给 eval 函数传递第 2 个参数，一个定义了该表达式求值时名称空间的字典. 我们测试下，给函数传递个空字典:

```
>>> print eval("__import__('os').remove('file')",
{})

Traceback (innermost last):

  File "<stdin>", line 1, in ?

  File "<string>", line 0, in ?

os.error: (2, 'No such file or directory')
```

呃.... 我们还是得到了个 *os.error* 异常.

这是因为 Python 在求值前会检查这个字典, 如果没有发现名称为 __builtins__ 的变量(复数形式), 它就会添加一个:

```
>>> namespace = {}
```

```
>>> print eval("__import__('os').remove('file')",
namespace)
```

```
Traceback (innermost last):

  File "<stdin>", line 1, in ?

  File "<string>", line 0, in ?

os.error: (2, 'No such file or directory')

>>> namespace.keys()

['__builtins__']
```

如果你打印这个 namespace 的内容，你会发现里边有所有的内建函数.

> [!Feather 注: 如果我 RP 不错的话，添加的这个_ _builtins_ _就是当前的_ _builtins_ _]

我们注意到了如果这个变量存在，Python 就不会去添加默认的，那么我们的解决方法也来了，为传递的字典参数加入一个 `_ _builtins_ _` 项即可. 如 Example 1-19 所示.

1.2.5.3. Example 1-19. 安全地使用 eval 函数求值

```
File: builtin-eval-example-3.py



print eval("_ _import_ _('os').getcwd()", {})
```

```
print eval("_ _import_ _('os').remove('file')", {"_ _builtins_ _": {}})
```

```


/home/fredrik/librarybook

Traceback (innermost last):

  File "builtin-eval-example-3.py", line 2, in ?

  File "<string>", line 0, in ?

NameError: _ _import_ _
```

即使这样，你仍然无法避免针对 CPU 和内存资源的攻击.（比如，形如 `eval("'*'*1000000*2*2*2*2*2*2*2*2")` 的语句在执行后会使你的程序耗尽系统资源）.

1.2.6. 编译和执行代码

`eval` 函数只针对简单的表达式. 如果要处理大块的代码，你应该使用 `compile` 和 `exec` 函数（如 Example 1-20 所示）.

1.2.6.1. Example 1-20. 使用 compile 函数检查语法

```
File: builtin-compile-example-1.py



NAME = "script.py"
```

```

```

```
BODY = """
prnt 'owl-stretching time'
"""



try:
    compile(BODY, NAME, "exec")
except SyntaxError, v:
    print "syntax error:", v, "in", NAME
```

```
# syntax error: invalid syntax in script.py
```

成功执行后，compile 函数会返回一个代码对象，你可以使用 exec 语句执行它，参见 Example 1-21 .

1.2.6.2. Example 1-21. 执行已编译的代码

```
File: builtin-compile-example-2.py

BODY = """
print 'the ant, an introduction'
"""

code = compile(BODY, "<script>", "exec")

print code

exec code
```

```
<code object ? at 8c6be0, file "<script>", line 0>

the ant, an introduction
```

使用 Example 1-22 中的类可以在程序执行时实时地生成代码. `write` 方法用于添加代码, `indent` 和 `dedent` 方法用于控制缩进结构. 其他部分交给类来处理.

1.2.6.3. Example 1-22. 简单的代码生成工具

```
File: builtin-compile-example-3.py
```

```
import sys, string
```

```
class CodeGeneratorBackend:

    "Simple code generator for Python"


    def begin(self, tab="\t"):

        self.code = []

        self.tab = tab

        self.level = 0
```

```
    def end(self):

        self.code.append("") # make sure there's a
newline at the end

        return compile(string.join(self.code,
"\n"), "<code>", "exec")



    def write(self, string):

        self.code.append(self.tab * self.level +
string)
```

```

```

```
    def indent(self):

        self.level = self.level + 1

        # in 2.0 and later, this can be written as:
self.level += 1



    def dedent(self):

        if self.level == 0:

            raise SyntaxError, "internal error in
code generator"

        self.level = self.level - 1
```

```
        # or: self.level -= 1



#

# try it out!



c = CodeGeneratorBackend()

c.begin()

c.write("for i in range(5):")
```

```
c.indent()
```

```
c.write("print 'code generation made easy!'")

c.dedent()

exec c.end()



code generation made easy!

code generation made easy!

code generation made easy!

code generation made easy!
```

```
code generation made easy!
```

Python 还提供了 `execfile` 函数, 一个从文件加载代码, 编译代码, 执行代码的快捷方式. Example 1-23 简单地展示了如何使用这个函数.

1.2.6.4. Example 1-23. 使用 execfile 函数

```
File: builtin-execfile-example-1.py



execfile("hello.py")
```

```
def EXECFILE(filename, locals=None, globals=None):
```

```
    exec compile(open(filename).read(), filename,
"exec") in locals, globals



EXECFILE("hello.py")



hello again, and welcome to the show

hello again, and welcome to the show
```

Example 1-24 中的代码是 Example 1-23 中使用的 hello.py 文件.

1.2.6.5. Example 1-24. hello.py 脚本

```
File: hello.py



print "hello again, and welcome to the show"
```

1.2.7. 从 _ _builtin_ _ 模块重载函数

因为 Python 在检查局部名称空间和模块名称空间前不会检查内建函数, 所以有时候你可能要显式地引用 _ _builtin_ _ 模块. 例如 Example 1-25 重载了内建的 open 函数. 这时候要想使用原来的 open 函数, 就需要脚本显式地指明模块名称.

1.2.7.1. Example 1-25. 显式地访问 _ _builtin_ _ 模块中的函数

```
File: builtin-open-example-1.py



def open(filename, mode="rb"):

    import _ _builtin_ _

    file = _ _builtin_ _.open(filename, mode)

    if file.read(5) not in("GIF87", "GIF89"):

        raise IOError, "not a GIF file"

    file.seek(0)

    return file
```

```
fp = open("samples/sample.gif")

print len(fp.read()), "bytes"



fp = open("samples/sample.jpg")

print len(fp.read()), "bytes"
```

```
3565 bytes
```

```
Traceback (innermost last):

  File "builtin-open-example-1.py", line 12, in ?

  File "builtin-open-example-1.py", line 5, in open

IOError: not a GIF file
```

[!Feather 注: 明白这个 open()函数是干什么的么? 检查一个文件是否是 GIF 文件,

一般如这类的图片格式都在文件开头有默认的格式.

另外打开文件推荐使用 file()而不是 open(), 虽然暂时没有区别]

1.3. exceptions 模块

exceptions 模块提供了标准异常的层次结构. Python 启动的时候会自动导入这个模块, 并且将它加入到 __builtin__ 模块中. 也就是说, 一般不需要手动导入这个模块.

在 1.5.2 版本时它是一个普通模块, 2.0 以及以后版本成为内建模块.

该模块定义了以下标准异常:

- *Exception* 是所有异常的基类. 强烈建议(但不是必须)自定义的异常异常也继承这个类.
- *SystemExit(Exception)* 由 sys.exit 函数引发. 如果它在最顶层没有被 try-except 语句捕获, 那么解释器将直接关闭而不会显示任何跟踪返回信息.
- *StandardError(Exception)* 是所有内建异常的基类(除 *SystemExit* 外).
- *KeyboardInterrupt(StandardError)* 在用户按下 Control-C(或其他打断按键)后 被引发. 如果它可能会在你使用 "捕获所有" 的 try-except 语句时导致奇怪的问题.
- *ImportError(StandardError)* 在 Python 导入模块失败时被引发.
- *EnvironmentError* 作为所有解释器环境引发异常的基类. (也就是说, 这些异常一般不是由于程序 bug 引起).
- *IOError(EnvironmentError)* 用于标记 I/O 相关错误.
- *OSError(EnvironmentError)* 用于标记 os 模块引起的错误.
- *WindowsError(OSError)* 用于标记 os 模块中 Windows 相关错误.
- *NameError(StandardError)* 在 Python 查找全局或局部名称失败时被引发.
- *UnboundLocalError(NameError)*, 当一个局部变量还没有赋值就被使用时, 会引发这个异常. 这个异常只有在 2.0 及之后的版本有; 早期版本只会引发一个普通的 *NameError* .
- *AttributeError(StandardError)*, 当 Python 寻找(或赋值)给一个实例属性, 方法, 模块功能或其它有效的命名失败时, 会引发这个异常.
- *SyntaxError(StandardError)*, 当解释器在编译时遇到语法错误, 这个异常就被引发.
- (2.0 及以后版本) *IndentationError(SyntaxError)* 在遇到非法的缩进时被引发. 该异常只用于 2.0 及以后版本, 之前版本会引发一个 *SyntaxError* 异常.
- (2.0 及以后版本) *TabError(IndentationError)*, 当使用 -tt 选项检查不一致缩进时有可能被引发. 该异常只用于 2.0 及以后版本, 之前版本会引发一个 *SyntaxError* 异常.
- *TypeError(StandardError)*, 当给定类型的对象不支持一个操作时被引发.
- *AssertionError(StandardError)* 在 assert 语句失败时被引发(即表达式为 false 时).
- *LookupError(StandardError)* 作为序列或字典没有包含给定索引或键时所引发异常的基类.

- *IndexError(LookupError)*，当序列对象使用给定索引数索引失败时(不存在索引对应对象)引发该异常.
- *KeyError(LookupError)* 当字典对象使用给定索引索引失败时(不存在索引对应对象)引发该异常.
- *ArithmeticError(StandardError)* 作为数学计算相关异常的基类.
- *OverflowError(ArithmeticError)* 在操作溢出时被引发(例如当一个整数太大，导致不能符合给定类型).
- *ZeroDivisionError(ArithmeticError)*，当你尝试用 0 除某个数时被引发.
- *FloatingPointError(ArithmeticError)*，当浮点数操作失败时被引发.
- *ValueError(StandardError)*，当一个参数类型正确但值不合法时被引发.
- (2.0 及以后版本) *UnicodeError(ValueError)*，Unicode 字符串类型相关异常. 只使用在 2.0 及以后版本.
- *RuntimeError(StandardError)*，当出现运行时问题时引发，包括在限制模式下尝试访问外部内容，未知的硬件问题等等.
- *NotImplementedError(RuntimeError)*，用于标记未实现的函数，或无效的方法.
- *SystemError(StandardError)*，解释器内部错误. 该异常值会包含更多的细节 (经常会是一些深层次的东西，比如 "eval_code2: NULL globals"). 这本书的作者编了 5 年程序都没见过这个错误. (想必是没有用 `raise SystemError`).
- *MemoryError(StandardError)*，当解释器耗尽内存时会引发该异常. 注意只有在底层内存分配抱怨时这个异常才会发生；如果是在你的旧机器上，这个异常发生之前系统会陷入混乱的内存交换中.

你可以创建自己的异常类. 只需要继承内建的 *Exception* 类(或者它的任意一个合适的子类)即可，有需要时可以再重载它的 `__str__` 方法. Example 1-26 展示了如何使用 `exceptions` 模块.

1.3.0.1. Example 1-26. 使用 exceptions 模块

```
File: exceptions-example-1.py



# python imports this module by itself, so the
following

# line isn't really needed
```

```
# python 会自动导入该模块, 所以以下这行是不必要的
```

```
# import exceptions
```

```


class HTTPError(Exception):

    # indicates an HTTP protocol error

    def _ _init_ _(self, url, errcode, errmsg):

        self.url = url

        self.errcode = errcode

        self.errmsg = errmsg

    def _ _str_ _(self):

        return (

            "<HTTPError for %s: %s %s>" %

            (self.url, self.errcode, self.errmsg)

            )



try:
```

```
    raise HTTPError("http://www.python.org/foo",
200, "Not Found")

except HTTPError, error:
```

```
    print "url", "=>", error.url
```

```
    print "errcode", "=>", error.errcode

    print "errmsg", "=>", error.errmsg

    raise # reraise exception
```

```
url => http://www.python.org/foo

errcode => 200

errmsg => Not Found

Traceback (innermost last):

  File "exceptions-example-1", line 16, in ?

HTTPError: <HTTPError for
http://www.python.org/foo: 200 Not Found>
```

1.4. os 模块

这个模块中的大部分函数通过对应平台相关模块实现，比如 posix 和 nt. os 模块会在第一次导入的时候自动加载合适的执行模块.

1.4.1. 处理文件

内建的 open / file 函数用于创建，打开和编辑文件，如 Example 1-27 所示. 而 os 模块提供了重命名和删除文件所需的函数.

1.4.1.1. Example 1-27. 使用 os 模块重命名和删除文件

```
File: os-example-3.py



import os

import string



def replace(file, search_for, replace_with):

    # replace strings in a text file



    back = os.path.splitext(file)[0] + ".bak"

    temp = os.path.splitext(file)[0] + ".tmp"



    try:

        # remove old temp file, if any
```

```
            os.remove(temp)
```

```
        except os.error:
```

```
            pass



        fi = open(file)

        fo = open(temp, "w")



        for s in fi.readlines():

            fo.write(string.replace(s, search_for,
    replace_with))



        fi.close()

        fo.close()



        try:

            # remove old backup file, if any

            os.remove(back)
```

```
    except os.error:

        pass
```

```

```

```
    # rename original to backup...

    os.rename(file, back)



    # ...and temporary to original

    os.rename(temp, file)



#

# try it out!



file = "samples/sample.txt"



replace(file, "hello", "tjena")

replace(file, "tjena", "hello")
```

1.4.2. 处理目录

os 模块也包含了一些用于目录处理的函数.

listdir 函数返回给定目录中所有文件名(包括目录名)组成的列表, 如 Example 1-28 所示. 而 Unix 和 Windows 中使用的当前目录和父目录标记(. 和 ..)不包含在此列表中.

1.4.2.1. Example 1-28. 使用 os 列出目录下的文件

```
File: os-example-5.py



import os



for file in os.listdir("samples"):

    print file
```

```
sample.au

sample.jpg

sample.wav

...
```

getcwd 和 chdir 函数分别用于获得和改变当前工作目录. 如 Example 1-29 所示.

1.4.2.2. Example 1-29. 使用 os 模块改变当前工作目录

```
File: os-example-4.py



import os



# where are we?

cwd = os.getcwd()

print "1", cwd



# go down

os.chdir("samples")

print "2", os.getcwd()



# go back up

os.chdir(os.pardir)

print "3", os.getcwd()



1 /ematter/librarybook
```

```
2 /ematter/librarybook/samples

3 /ematter/librarybook
```

makedirs 和 removedirs 函数用于创建或删除目录层，如 Example 1-30 所示.

1.4.2.3. Example 1-30. 使用 os 模块创建/删除多个目录级

```
File: os-example-6.py



import os



os.makedirs("test/multiple/levels")



fp = open("test/multiple/levels/file", "w")

fp.write("inspector praline")

fp.close()



# remove the file

os.remove("test/multiple/levels/file")
```

```
# and all empty directories above it

os.removedirs("test/multiple/levels")
```

removedirs 函数会删除所给路径中最后一个目录下所有的空目录. 而 mkdir 和 rmdir 函数只能处理单个目录级. 如 Example 1-31 所示.

1.4.2.4. Example 1-31. 使用 os 模块创建/删除目录

```
File: os-example-7.py

import os

os.mkdir("test")

os.rmdir("test")

os.rmdir("samples") # this will fail

Traceback (innermost last):

  File "os-example-7", line 6, in ?
```

```
OSError: [Errno 41] Directory not empty: 'samples'
```

如果需要删除非空目录, 你可以使用 `shutil` 模块中的 `rmtree` 函数.

1.4.3. 处理文件属性

`stat` 函数可以用来获取一个存在文件的信息, 如 Example 1-32 所示. 它返回一个类元组对象(stat_result 对象, 包含 10 个元素), 依次是 st_mode (权限模式), st_ino (inode number), st_dev (device), st_nlink (number of hard links), st_uid (所有者用户 ID), st_gid (所有者所在组 ID), st_size (文件大小, 字节), st_atime (最近一次访问时间), st_mtime (最近修改时间), st_ctime (平台相关; Unix 下的最近一次元数据/metadata 修改时间, 或者 Windows 下的创建时间) - 以上项目也可作为属性访问.

```
[!Feather 注: 原文为 9 元元组. 另,返回对象并非元组类型,为 struct.]
```

1.4.3.1. Example 1-32. 使用 os 模块获取文件属性

```
File: os-example-1.py



import os

import time



file = "samples/sample.jpg"
```

```
def dump(st):

    mode, ino, dev, nlink, uid, gid, size, atime,
mtime, ctime = st

    print "- size:", size, "bytes"

    print "- owner:", uid, gid

    print "- created:", time.ctime(ctime)

    print "- last accessed:", time.ctime(atime)

    print "- last modified:", time.ctime(mtime)

    print "- mode:", oct(mode)

    print "- inode/dev:", ino, dev



#

# get stats for a filename



st = os.stat(file)



print "stat", file

dump(st)
```

```
print



#

# get stats for an open file



fp = open(file)



st = os.fstat(fp.fileno())



print "fstat", file

dump(st)



stat samples/sample.jpg

- size: 4762 bytes

- owner: 0 0

- created: Tue Sep 07 22:45:58 1999

- last accessed: Sun Sep 19 00:00:00 1999
```

```
- last modified: Sun May 19 01:42:16 1996

- mode: 0100666

- inode/dev: 0 2



fstat samples/sample.jpg

- size: 4762 bytes

- owner: 0 0

- created: Tue Sep 07 22:45:58 1999

- last accessed: Sun Sep 19 00:00:00 1999

- last modified: Sun May 19 01:42:16 1996

- mode: 0100666

- inode/dev: 0 0
```

返回对象中有些属性在非 Unix 平台下是无意义的, 比如 (st_inode , st_dev)为 Unix 下的为每个文件提供了唯一标识, 但在其他平台可能为任意无意义数据 .

stat 模块包含了很多可以处理该返回对象的常量及函数. 下面的代码展示了其中的一些.

可以使用 chmod 和 utime 函数修改文件的权限模式和时间属性，如 Example 1-33 所示.

1.4.3.2. Example 1-33. 使用 os 模块修改文件的权限和时间戳

```
File: os-example-2.py



import os

import stat, time



infile = "samples/sample.jpg"

outfile = "out.jpg"



# copy contents

fi = open(infile, "rb")

fo = open(outfile, "wb")



while 1:

    s = fi.read(10000)

    if not s:

        break

    fo.write(s)
```

```
fi.close()

fo.close()



# copy mode and timestamp

st = os.stat(infile)

os.chmod(outfile, stat.S_IMODE(st[stat.ST_MODE]))

os.utime(outfile, (st[stat.ST_ATIME],
st[stat.ST_MTIME]))



print "original", "=>"

print "mode", oct(stat.S_IMODE(st[stat.ST_MODE]))

print "atime", time.ctime(st[stat.ST_ATIME])

print "mtime", time.ctime(st[stat.ST_MTIME])



print "copy", "=>"

st = os.stat(outfile)
```

```
print "mode", oct(stat.S_IMODE(st[stat.ST_MODE]))

print "atime", time.ctime(st[stat.ST_ATIME])

print "mtime", time.ctime(st[stat.ST_MTIME])



original =>

mode 0666

atime Thu Oct 14 15:15:50 1999

mtime Mon Nov 13 15:42:36 1995

copy =>

mode 0666

atime Thu Oct 14 15:15:50 1999

mtime Mon Nov 13 15:42:36 1995
```

1.4.4. 处理进程

system 函数在当前进程下执行一个新命令，并等待它完成，如 Example 1-34 所示.

1.4.4.1. Example 1-34. 使用 os 执行操作系统命令

```
File: os-example-8.py
```

```
import os



if os.name == "nt":

    command = "dir"

else:

    command = "ls -l"



os.system(command)
```

```
-rwxrw-r--   1 effbot  effbot        76 Oct  9
14:17 README

-rwxrw-r--   1 effbot  effbot      1727 Oct  7
19:00 SimpleAsyncHTTP.py

-rwxrw-r--   1 effbot  effbot       314 Oct  7
20:29 aifc-example-1.py

-rwxrw-r--   1 effbot  effbot       259 Oct  7
20:38 anydbm-example-1.py

...
```

命令通过操作系统的标准 shell 执行，并返回 shell 的退出状态. 需要注意的是在 Windows 95/98 下，shell 通常是 command.com，它的推出状态总是 0.

由于 11os.system11 直接将命令传递给 shell，所以如果你不检查传入参数的时候会很危险（比如命令 os.system("viewer %s" % file)，将 file 变量设置为 "sample.jpg; rm -rf $HOME" ....). 如果不确定参数的安全性，那么最好使用 exec 或 spawn 代替(稍后介绍).

exec 函数会使用新进程替换当前进程(或者说是"转到进程"). 在 Example 1-35 中，字符串 "goodbye" 永远不会被打印.

1.4.4.2. Example 1-35. 使用 os 模块启动新进程

```
File: os-exec-example-1.py



import os

import sys



program = "python"

arguments = ["hello.py"]



print os.execvp(program, (program,) +
tuple(arguments))

print "goodbye"
```

```
```

```
hello again, and welcome to the show
```

Python 提供了很多表现不同的 `exec` 函数. Example 1-35 使用的是 `execvp` 函数, 它会从标准路径搜索执行程序, 把第二个参数(元组)作为单独的参数传递给程序, 并使用当前的环境变量来运行程序. 其他七个同类型函数请参阅 *Python Library Reference* .

在 Unix 环境下, 你可以通过组合使用 `exec` , `fork` 以及 `wait` 函数来从当前程序调用另一个程序, 如 Example 1-36 所示. `fork` 函数复制当前进程, `wait` 函数会等待一个子进程执行结束.

1.4.4.3. Example 1-36. 使用 os 模块调用其他程序 (Unix)

```
File: os-exec-example-2.py



import os

import sys



def run(program, *args):

    pid = os.fork()

    if not pid:

        os.execvp(program, (program,) +  args)

    return os.wait()[0]


```

```
run("python", "hello.py")



print "goodbye"



hello again, and welcome to the show

goodbye
```

fork 函数在子进程返回中返回 0 (这个进程首先从 fork 返回值), 在父进程中返回一个非 0 的进程标识符(子进程的 PID). 也就是说, 只有当我们处于子进程的时候 "not pid" 才为真.

fork 和 wait 函数在 Windows 上是不可用的, 但是你可以使用 spawn 函数, 如 Example 1-37 所示. 不过, spawn 不会沿着路径搜索可执行文件, 你必须自己处理好这些.

1.4.4.4. Example 1-37. 使用 os 模块调用其他程序 (Windows)

```
File: os-spawn-example-1.py



import os
```

```
import string
```

```


def run(program, *args):
```

```
    # find executable

    for path in string.split(os.environ["PATH"],
os.pathsep):

        file = os.path.join(path, program) + ".exe"

        try:

            return os.spawnv(os.P_WAIT, file,
(file,) + args)

        except os.error:

            pass

    raise os.error, "cannot find executable"



run("python", "hello.py")



print "goodbye"
```

```
hello again, and welcome to the show
```

```
goodbye
```

spawn 函数还可用于在后台运行一个程序. Example 1-38 给 run 函数添加了一个可选的 mode 参数; 当设置为 os.P_NOWAIT 时, 这个脚本不会等待子程序结束, 默认值 os.P_WAIT 时 spawn 会等待子进程结束.

其它的标志常量还有 os.P_OVERLAY ,它使得 spawn 的行为和 exec 类似, 以及 os.P_DETACH , 它在后台运行子进程, 与当前控制台和键盘焦点隔离.

1.4.4.5. Example 1-38. 使用 os 模块在后台执行程序 (Windows)

```
File: os-spawn-example-2.py



import os

import string



def run(program, *args, **kw):

    # find executable

    mode = kw.get("mode", os.P_WAIT)

    for path in string.split(os.environ["PATH"],
os.pathsep):

        file = os.path.join(path, program) + ".exe"
```

```
        try:
```

```
            return os.spawnv(mode, file, (file,) +
args)
```

```
        except os.error:
            pass
    raise os.error, "cannot find executable"

run("python", "hello.py", mode=os.P_NOWAIT)
print "goodbye"
```

```
goodbye
hello again, and welcome to the show
```

Example 1-39 提供了一个在 Unix 和 Windows 平台上通用的 spawn 方法.

1.4.4.6. Example 1-39. 使用 spawn 或 fork/exec 调用其他程序

```
File: os-spawn-example-3.py

import os
```

```
import string
```

```
```

```
if os.name in ("nt", "dos"):
    exefile = ".exe"
else:
    exefile = ""


def spawn(program, *args):
    try:
        # possible 2.0 shortcut!
        return os.spawnvp(program, (program,) +
args)
    except AttributeError:
        pass
    try:
        spawnv = os.spawnv
    except AttributeError:
```

```
        # assume it's unix
```

```
        pid = os.fork()

        if not pid:

            os.execvp(program, (program,) + args)

        return os.wait()[0]

    else:

        # got spawnv but no spawnp: go look for an
executable

        for path in
string.split(os.environ["PATH"], os.pathsep):

            file = os.path.join(path, program) +
exefile

            try:

                return spawnv(os.P_WAIT, file,
(file,) + args)

            except os.error:

                pass

        raise IOError, "cannot find executable"



#
```

```
# try it out!

spawn("python", "hello.py")

print "goodbye"
```

```
hello again, and welcome to the show

goodbye
```

Example 1-39 首先尝试调用 spawnvp 函数. 如果该函数不存在 (一些版本/平台没有这个函数), 它将继续查找一个名为 spawnv 的函数并且开始查找程序路径. 作为最后的选择, 它会调用 exec 和 fork 函数完成工作.

1.4.5. 处理守护进程(Daemon Processes)

Unix 系统中, 你可以使用 fork 函数把当前进程转入后台(一个"守护者/daemon"). 一般来说, 你需要派生(fork off)一个当前进程的副本, 然后终止原进程, 如 Example 1-40 所示.

1.4.5.1. Example 1-40. 使用 os 模块使脚本作为守护执行 (Unix)

```
File: os-example-14.py

import os
```

```
import time



pid = os.fork()

if pid:

    os._exit(0) # kill original



print "daemon started"

time.sleep(10)

print "daemon terminated"
```

需要创建一个真正的后台程序稍微有点复杂, 首先调用 `setpgrp` 函数创建一个 "进程组首领/process group leader". 否则, 向无关进程组发送的信号(同时)会引起守护进程的问题:

```
os.setpgrp()
```

为了确保守护进程创建的文件能够获得程序指定的 mode flags(权限模式标记?), 最好删除 user mode mask:

```
os.umask(0)
```

然后, 你应该重定向 *stdout/stderr* 文件, 而不能只是简单地关闭它们(如果你的程序需要 `stdout` 或 `stderr` 写入内容的时候, 可能会出现意想不到的问题).

```
class NullDevice:

    def write(self, s):

        pass

sys.stdin.close()

sys.stdout = NullDevice()

sys.stderr = NullDevice()
```

换言之，由于 Python 的 `print` 和 C 中的 `printf/fprintf` 在设备(device) 没有连接后不会关闭你的程序，此时守护进程中的 `sys.stdout.write()` 会抛出一个 *IOError* 异常，而你的程序依然在后台运行的很好....

另外，先前例子中的 `_exit` 函数会终止当前进程. 而 `sys.exit` 不同，如果调用者(caller) 捕获了 *SystemExit* 异常，程序仍然会继续执行. 如 Example 1-41 所示.

1.4.5.2. Example 1-41. 使用 os 模块终止当前进程

```
File: os-example-9.py



import os
```

```
import sys
```

```


try:
```

```
    sys.exit(1)

except SystemExit, value:

    print "caught exit(%s)" % value



try:

    os._exit(2)

except SystemExit, value:

    print "caught exit(%s)" % value



print "bye!"



caught exit(1)
```

1.5. os.path 模块

os.path 模块包含了各种处理长文件名(路径名)的函数. 先导入 (import) os 模块, 然后就可以以 os.path 访问该模块.

1.5.1. 处理文件名

os.path 模块包含了许多与平台无关的处理长文件名的函数. 也就是说, 你不需要处理前后斜杠, 冒号等. 我们可以看看 Example 1-42 中的样例代码.

1.5.1.1. Example 1-42. 使用 os.path 模块处理文件名

```
File: os-path-example-1.py



import os



filename = "my/little/pony"



print "using", os.name, "..."

print "split", "=>", os.path.split(filename)

print "splitext", "=>", os.path.splitext(filename)

print "dirname", "=>", os.path.dirname(filename)

print "basename", "=>", os.path.basename(filename)
```

```
print "join", "=>",
os.path.join(os.path.dirname(filename),
```

```
os.path.basename(filename))



using nt ...
```

```
split => ('my/little', 'pony')

splitext => ('my/little/pony', '')

dirname => my/little

basename => pony

join => my/little\pony
```

注意这里的 `split` 只分割出最后一项(不带斜杠).

`os.path` 模块中还有许多函数允许你简单快速地获知文件名的一些特征，如 Example 1-43 所示。

1.5.1.2. Example 1-43. 使用 os.path 模块检查文件名的特征

```
File: os-path-example-2.py



import os
```

```
FILES = (
```

```
    os.curdir,

    "/",

    "file",
```

```
        "/file",
        "samples",
        "samples/sample.jpg",
        "directory/file",
        "../directory/file",
        "/directory/file"
        )


    for file in FILES:
        print file, "=>",
        if os.path.exists(file):
            print "EXISTS",
        if os.path.isabs(file):
            print "ISABS",
        if os.path.isdir(file):
            print "ISDIR",
        if os.path.isfile(file):
```

```
        print "ISFILE",
    if os.path.islink(file):
        print "ISLINK",
    if os.path.ismount(file):
        print "ISMOUNT",
    print
```

```
. => EXISTS ISDIR
/ => EXISTS ISABS ISDIR ISMOUNT
file =>
/file => ISABS
samples => EXISTS ISDIR
samples/sample.jpg => EXISTS ISFILE
```

```
directory/file =>
```

```
../directory/file =>
/directory/file => ISABS
```

expanduser 函数以与大部分 Unix shell 相同的方式处理用户名快捷符号(~, 不过在 Windows 下工作不正常), 如 Example 1-44 所示.

1.5.1.3. Example 1-44. 使用 os.path 模块将用户名插入到文件名

```
File: os-path-expanduser-example-1.py



import os



print os.path.expanduser("~/.pythonrc")



# /home/effbot/.pythonrc
```

expandvars 函数将文件名中的环境变量替换为对应值, 如 Example 1-45 所示.

1.5.1.4. Example 1-45. 使用 os.path 替换文件名中的环境变量

```
File: os-path-expandvars-example-1.py
```

```

```

```
import os



os.environ["USER"] = "user"
```

```
print os.path.expandvars("/home/$USER/config")

print os.path.expandvars("$USER/folders")
```

```
/home/user/config

user/folders
```

1.5.2. 搜索文件系统

walk 函数会帮你找出一个目录树下的所有文件 (如 Example 1-46 所示). 它的参数依次是目录名, 回调函数, 以及传递给回调函数的数据对象.

1.5.2.1. Example 1-46. 使用 os.path 搜索文件系统

```
File: os-path-walk-example-1.py

import os
```

```

def callback(arg, directory, files):

    for file in files:
```

```
        print os.path.join(directory, file),
repr(arg)



os.path.walk(".", callback, "secret message")



./aifc-example-1.py 'secret message'

./anydbm-example-1.py 'secret message'

./array-example-1.py 'secret message'

...

./samples 'secret message'

./samples/sample.jpg 'secret message'

./samples/sample.txt 'secret message'

./samples/sample.zip 'secret message'

./samples/articles 'secret message'

./samples/articles/article-1.txt 'secret message'

./samples/articles/article-2.txt 'secret message'

...
```

walk 函数的接口多少有点晦涩（也许只是对我个人而言，我总是记不住参数的顺序）. Example 1-47 中展示的 index 函数会返回一个文件名列表，你可以直接使用 for-in 循环处理文件.

1.5.2.2. Example 1-47. 使用 os.listdir 搜索文件系统

```
File: os-path-walk-example-2.py



import os



def index(directory):

    # like os.listdir, but traverses directory trees

    stack = [directory]

    files = []

    while stack:

        directory = stack.pop()

        for file in os.listdir(directory):

            fullname = os.path.join(directory,
file)

            files.append(fullname)
```

```
            if os.path.isdir(fullname) and not
os.path.islink(fullname):

                stack.append(fullname)

    return files



for file in index(".."):

    print file



.\aifc-example-1.py

.\anydbm-example-1.py

.\array-example-1.py

...
```

如果你不想列出所有的文件（基于性能或者是内存的考虑），Example 1-48 展示了另一种方法. 这里 *DirectoryWalker* 类的行为与序列对象相似, 一次返回一个文件. (generator?)

1.5.2.3. Example 1-48. 使用 DirectoryWalker 搜索文件系统

```
File: os-path-walk-example-3.py



import os
```

```
class DirectoryWalker:

    # a forward iterator that traverses a directory
tree



    def _ _init_ _(self, directory):

        self.stack = [directory]

        self.files = []

        self.index = 0



    def _ _getitem_ _(self, index):

        while 1:

            try:

                file = self.files[self.index]

                self.index = self.index + 1

            except IndexError:

                # pop next directory from stack
```

```
                self.directory = self.stack.pop()

                self.files =
os.listdir(self.directory)

                self.index = 0

            else:

                # got a filename

                fullname =
os.path.join(self.directory, file)

                if os.path.isdir(fullname) and not
os.path.islink(fullname):

                    self.stack.append(fullname)

                return fullname



for file in DirectoryWalker("."):

    print file
```

```
.\aifc-example-1.py
```

```
.\anydbm-example-1.py
```

```
.\array-example-1.py
```

```
...
```

注意 *DirectoryWalker* 类并不检查传递给 `__getitem__` 方法的索引值. 这意味着如果你越界访问序列成员(索引数字过大)的话, 这个类将不能正常工作.

最后, 如果你需要处理文件大小和时间戳, Example 1-49 给出了一个类, 它返回文件名和它的 `os.stat` 属性(一个元组). 这个版本在每个文件上都能节省一次或两次 `stat` 调用(`os.path.isdir` 和 `os.path.islink` 内部都使用了 `stat`), 并且在一些平台上运行很快.

1.5.2.4. Example 1-49. 使用 DirectoryStatWalker 搜索文件系统

```
File: os-path-walk-example-4.py



import os, stat



class DirectoryStatWalker:

    # a forward iterator that traverses a directory
tree, and

    # returns the filename and additional file
information
```

```

    def __init__(self, directory):

        self.stack = [directory]
```

```
        self.files = []
        self.index = 0

    def __getitem__(self, index):
        while 1:
            try:
                file = self.files[self.index]
                self.index = self.index + 1
            except IndexError:
                # pop next directory from stack
                self.directory = self.stack.pop()
                self.files =
os.listdir(self.directory)
                self.index = 0
            else:
                # got a filename
                fullname =
os.path.join(self.directory, file)
```

```
                    st = os.stat(fullname)

                    mode = st[stat.ST_MODE]

                    if stat.S_ISDIR(mode) and not
stat.S_ISLNK(mode):

                        self.stack.append(fullname)

                    return fullname, st



for file, st in DirectoryStatWalker("."):

    print file, st[stat.ST_SIZE]



.\aifc-example-1.py 336

.\anydbm-example-1.py 244

.\array-example-1.py 526
```

1.6. stat 模块

Example 1-50 展示了 stat 模块的基本用法, 这个模块包含了一些 os.stat 函数中可用的常量和测试函数.

1.6.0.1. Example 1-50. Using the stat Module

```
File: stat-example-1.py



import stat

import os, time



st = os.stat("samples/sample.txt")



print "mode", "=>",
oct(stat.S_IMODE(st[stat.ST_MODE]))



print "type", "=>",

if stat.S_ISDIR(st[stat.ST_MODE]):

    print "DIRECTORY",

if stat.S_ISREG(st[stat.ST_MODE]):

    print "REGULAR",
```

```
if stat.S_ISLNK(st[stat.ST_MODE]):
```

```
    print "LINK",
```

```
print



print "size", "=>", st[stat.ST_SIZE]



print "last accessed", "=>",
time.ctime(st[stat.ST_ATIME])

print "last modified", "=>",
time.ctime(st[stat.ST_MTIME])

print "inode changed", "=>",
time.ctime(st[stat.ST_CTIME])



mode => 0664

type => REGULAR

size => 305

last accessed => Sun Oct 10 22:12:30 1999

last modified => Sun Oct 10 18:39:37 1999

inode changed => Sun Oct 10 15:26:38 1999
```

1.7. string 模块

string 模块提供了一些用于处理字符串类型的函数，如 Example 1-51 所示.

1.7.0.1. Example 1-51. 使用 string 模块

```
File: string-example-1.py



import string



text = "Monty Python's Flying Circus"



print "upper", "=>", string.upper(text)

print "lower", "=>", string.lower(text)

print "split", "=>", string.split(text)

print "join", "=>",
string.join(string.split(text), "+")

print "replace", "=>", string.replace(text,
"Python", "Java")

print "find", "=>", string.find(text, "Python"),
string.find(text, "Java")

print "count", "=>", string.count(text, "n")
```

```
upper => MONTY PYTHON'S FLYING CIRCUS

lower => monty python's flying circus

split => ['Monty', "Python's", 'Flying', 'Circus']

join => Monty+Python's+Flying+Circus

replace => Monty Java's Flying Circus

find => 6 -1

count => 3
```

在 Python 1.5.2 以及更早版本中, `string` 使用 `strop` 中的函数来实现模块功能.

在 Python1.6 和后继版本，更多的字符串操作都可以作为字符串方法来访问, 如 Example 1-52 所示, `string` 模块中的许多函数只是对相对应字符串方法的封装.

1.7.0.2. Example 1-52. 使用字符串方法替代 string 模块函数

```
File: string-example-2.py



text = "Monty Python's Flying Circus"



print "upper", "=>", text.upper()
```

```
print "lower", "=>", text.lower()

print "split", "=>", text.split()

print "join", "=>", "+".join(text.split())

print "replace", "=>", text.replace("Python",
"Perl")

print "find", "=>", text.find("Python"),
text.find("Perl")

print "count", "=>", text.count("n")
```

```
upper => MONTY PYTHON'S FLYING CIRCUS

lower => monty python's flying circus

split => ['Monty', "Python's", 'Flying', 'Circus']

join => Monty+Python's+Flying+Circus

replace => Monty Perl's Flying Circus

find => 6 -1

count => 3
```

为了增强模块对字符的处理能力，除了字符串方法，`string` 模块还包含了类型转换函数用于把字符串转换为其他类型，（如 Example 1-53 所示）.

1.7.0.3. Example 1-53. 使用 string 模块将字符串转为数字

```
File: string-example-3.py



import string



print int("4711"),

print string.atoi("4711"),

print string.atoi("11147", 8), # octal 八进制

print string.atoi("1267", 16), # hexadecimal 十六
进制

print string.atoi("3mv", 36) # whatever...



print string.atoi("4711", 0),

print string.atoi("04711", 0),

print string.atoi("0x4711", 0)



print float("4711"),

print string.atof("1"),
```

```
print string.atof("1.23e5")



4711 4711 4711 4711 4711

4711 2505 18193

4711.0 1.0 123000.0
```

大多数情况下（特别是当你使用的是 1.6 及更高版本时），你可以使用 `int` 和 `float` 函数代替 `string` 模块中对应的函数。

`atoi` 函数可以接受可选的第二个参数，指定数基(number base)．如果数基为 0，那么函数将检查字符串的前几个字符来决定使用的数基：如果为 "0x," 数基将为 16（十六进制），如果为 "0," 则数基为 8（八进制）．默认数基值为 10（十进制），当你未传递参数时就使用这个值．

在 1.6 及以后版本中，`int` 函数和 `atoi` 一样可以接受第二个参数．与字符串版本函数不一样的是，`int` 和 `float` 可以接受 Unicode 字符串对象．

1.8. re 模块

"Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems."
- Jamie Zawinski, on comp.lang.emacs

`re` 模块提供了一系列功能强大的正则表达式（regular expression）工具，它们允许你快速检查给定字符串是否与给定的模式匹配（使用 `match` 函数），或者包含这个模式（使用 `search` 函数）．正则表达式是以紧凑(也很神秘)的语法写出的字符串模式．

`match` 尝试从字符串的起始匹配一个模式，如 Example 1-54 所示．如果模式匹配了某些内容（包括空字符串，如果模式允许的话），它将返回一个匹配对象．使用它的 `group` 方法可以找出匹配的内容．

1.8.0.1. Example 1-54. 使用 re 模块来匹配字符串

```
File: re-example-1.py
```

```
import re


text = "The Attila the Hun Show"


# a single character 单个字符

m = re.match(".", text)

if m: print repr("."), "=>", repr(m.group(0))


# any string of characters 任何字符串

m = re.match(".*", text)

if m: print repr(".*"), "=>", repr(m.group(0))


# a string of letters (at least one) 只包含字母的字符串(至少一个)

m = re.match("\w+", text)

if m: print repr("\w+"), "=>", repr(m.group(0))
```

```
# a string of digits 只包含数字的字符串

m = re.match("\d+", text)

if m: print repr("\d+"), "=>", repr(m.group(0))
```

```
 '.' => 'T'

'.*' => 'The Attila the Hun Show'

'\\w+' => 'The'
```

可以使用圆括号在模式中标记区域. 找到匹配后, group 方法可以抽取这些区域的内容，如 Example 1-55 所示. group(1) 会返回第一组的内容, group(2) 返回第二组的内容, 这样... 如果你传递多个组数给 group 函数, 它会返回一个元组.

1.8.0.2. Example 1-55. 使用 re 模块抽出匹配的子字符串

```
File: re-example-2.py
```

```

```

```
import re



text ="10/15/99"
```

```
m = re.match("(\d{2})/(\d{2})/(\d{2,4})", text)

if m:

    print m.group(1, 2, 3)
```

```
('10', '15', '99')
```

search 函数会在字符串内查找模式匹配，如 Example 1-56 所示. 它在所有可能的字符位置尝试匹配模式，从最左边开始，一旦找到匹配就返回一个匹配对象. 如果没有找到相应的匹配，就返回 *None* .

1.8.0.3. Example 1-56. 使用 re 模块搜索子字符串

```
File: re-example-3.py

import re
```

```
text = "Example 3: There is 1 date 10/25/95 in here!"
```

```
m = re.search("(\d{1,2})/(\d{1,2})/(\d{2,4})",
text)
```

```
print m.group(1), m.group(2), m.group(3)



month, day, year = m.group(1, 2, 3)

print month, day, year



date = m.group(0)

print date



10 25 95

10 25 95

10/25/95
```

Example 1-57 中展示了 sub 函数, 它可以使用另个字符串替代匹配模式.

1.8.0.4. Example 1-57. 使用 re 模块替换子字符串

```
File: re-example-4.py



import re
```

```
text = "you're no fun anymore..."



# literal replace (string.replace is faster)

# 文字替换 (string.replace 速度更快)

print re.sub("fun", "entertaining", text)



# collapse all non-letter sequences to a single dash

# 将所有非字母序列转换为一个"-"(dansh,破折号)

print re.sub("[^\w]+", "-", text)



# convert all words to beeps

# 将所有单词替换为 BEEP

print re.sub("\S+", "-BEEP-", text)
```

```

```

```
you're no entertaining anymore...

you-re-no-fun-anymore-
```

```
-BEEP- -BEEP- -BEEP- -BEEP-
```

你也可以通过回调（callback）函数使用 sub 来替换指定模式. Example 1-58 展示了如何预编译模式.

1.8.0.5. Example 1-58. 使用 re 模块替换字符串(通过回调函数)

```
File: re-example-5.py



import re

import string



text = "a line of text\\012another line of
text\\012etc..."



def octal(match):

    # replace octal code with corresponding ASCII
character

    # 使用对应 ASCII 字符替换八进制代码

    return chr(string.atoi(match.group(1), 8))



```

```
octal_pattern = re.compile(r"\\(\d\d\d)")



print text

print octal_pattern.sub(octal, text)



a line of text\012another line of text\012etc...

a line of text

another line of text

etc...
```

如果你不编译，re 模块会为你缓存一个编译后版本，所有的小脚本中，通常不需要编译正则表达式. Python1.5.2 中，缓存中可以容纳 20 个匹配模式，而在 2.0 中，缓存则可以容纳 100 个匹配模式.

最后，Example 1-59 用一个模式列表匹配一个字符串. 这些模式将会组合为一个模式，并预编译以节省时间.

1.8.0.6. Example 1-59. 使用 re 模块匹配多个模式中的一个

```
File: re-example-6.py
```

```

```

```
import re, string



```

```
def combined_pattern(patterns):

    p = re.compile(

        string.join(map(lambda x: "("+x+")",
patterns), "|")

        )

    def fixup(v, m=p.match,
r=range(0,len(patterns))):

        try:

            regs = m(v).regs

        except AttributeError:

            return None # no match, so m.regs will
fail

        else:

            for i in r:

                if regs[i+1] != (-1, -1):

                    return i
```

```
    return fixup
```

```

#
```

```
# try it out!



patterns = [

    r"\d+",

    r"abc\d{2,4}",

    r"p\w+"

]



p = combined_pattern(patterns)



print p("129391")

print p("abc800")

print p("abc1600")

print p("python")
```

```
print p("perl")
```

```
print p("tcl")
```

```
0

1

1

2

2

None
```

1.9. math 模块

math 模块实现了许多对浮点数的数学运算函数. 这些函数一般是对平台 C 库中同名函数的简单封装, 所以一般情况下, 不同平台下计算的结果可能稍微地有所不同, 有时候甚至有很大出入. Example 1-60 展示了如何使用 math 模块.

1.9.0.1. Example 1-60. 使用 math 模块

```
File: math-example-1.py
```

```
import math
```

```


print "e", "=>", math.e
```

```
print "pi", "=>", math.pi

print "hypot", "=>", math.hypot(3.0, 4.0)



# and many others...
```

```
e => 2.71828182846

pi => 3.14159265359

hypot => 5.0
```

完整函数列表请参阅 *Python Library Reference* .

1.10. cmath 模块

Example 1-61 所展示的 cmath 模块包含了一些用于复数运算的函数.

1.10.0.1. Example 1-61. 使用 cmath 模块

```
File: cmath-example-1.py
```

```

```

```
import cmath


```

```
print "pi", "=>", cmath.pi

print "sqrt(-1)", "=>", cmath.sqrt(-1)



pi => 3.14159265359

sqrt(-1) => 1j
```

完整函数列表请参阅 *Python Library Reference* .

1.11. operator 模块

operator 模块为 Python 提供了一个 "功能性" 的标准操作符接口. 当使用 map 以及 filter 一类的函数的时候, operator 模块中的函数可以替换一些 lambda 函式. 而且这些函数在一些喜欢写晦涩代码的程序员中很流行. Example 1-62 展示了 operator 模块的一般用法.

1.11.0.1. Example 1-62. 使用 operator 模块

```
File: operator-example-1.py
```

```
import operator
```

```


sequence = 1, 2, 4


```

```
print "add", "=>", reduce(operator.add, sequence)

print "sub", "=>", reduce(operator.sub, sequence)

print "mul", "=>", reduce(operator.mul, sequence)

print "concat", "=>", operator.concat("spam",
"egg")

print "repeat", "=>", operator.repeat("spam", 5)

print "getitem", "=>", operator.getitem(sequence,
2)

print "indexOf", "=>", operator.indexOf(sequence,
2)

print "sequenceIncludes", "=>",
operator.sequenceIncludes(sequence, 3)
```

```
add => 7

sub => -5

mul => 8
```

```
concat => spamegg
```

```
repeat => spamspamspamspamspam
```

```
getitem => 4

indexOf => 1

sequenceIncludes => 0
```

Example 1-63 展示了一些可以用于检查对象类型的 operator 函数.

1.11.0.2. Example 1-63. 使用 operator 模块检查类型

```
File: operator-example-2.py



import operator

import UserList



def dump(data):

    print type(data), "=>",

    if operator.isCallable(data):

        print "CALLABLE",
```

```
    if operator.isMappingType(data):
```

```
        print "MAPPING",
```

```
    if operator.isNumberType(data):
        print "NUMBER",
    if operator.isSequenceType(data):
        print "SEQUENCE",
    print


dump(0)
dump("string")
dump("string"[0])
dump([1, 2, 3])
dump((1, 2, 3))
dump({"a": 1})
dump(len) # function 函数
dump(UserList) # module 模块
dump(UserList.UserList) # class 类
dump(UserList.UserList()) # instance 实例
```

```
<type 'int'> => NUMBER

<type 'string'> => SEQUENCE

<type 'string'> => SEQUENCE

<type 'list'> => SEQUENCE

<type 'tuple'> => SEQUENCE

<type 'dictionary'> => MAPPING

<type 'builtin_function_or_method'> => CALLABLE

<type 'module'> =>

<type 'class'> => CALLABLE

<type 'instance'> => MAPPING NUMBER SEQUENCE
```

这里需要注意 operator 模块使用非常规的方法处理对象实例. 所以使用 isNumberType, isMappingType, 以及 isSequenceType 函数的时候要小心, 这很容易降低代码的扩展性.

同样需要注意的是一个字符串序列成员 (单个字符) 也是序列. 所以当在递归函数使用 isSequenceType 来截断对象树的时候, 别把普通字符串作为参数(或者是任何包含字符串的序列对象).

1.12. copy 模块

copy 模块包含两个函数, 用来拷贝对象, 如 Example 1-64 所示.

copy(object) => object 创建给定对象的 "浅/浅层(shallow)" 拷贝(copy). 这里 "浅/浅层(shallow)" 的意思是复制对象本身, 但当对象是一个容器 (container) 时, 它的成员仍然指向原来的成员对象.

1.12.0.1. Example 1-64. 使用 copy 模块复制对象

```
File: copy-example-1.py



import copy



a = [[1],[2],[3]]

b = copy.copy(a)



print "before", "=>"

print a

print b



# modify original

a[0][0] = 0

a[1] = None
```

```

```

```
print "after", "=>"
```

```
print a

print b



before =>

[[1], [2], [3]]

[[1], [2], [3]]

after =>

[[0], None, [3]]

[[0], [2], [3]]
```

你也可以使用[:]语句（完整切片）来对列表进行浅层复制，也可以使用 copy 方法复制字典.

相反地，deepcopy(object) => object 创建一个对象的深层拷贝(deepcopy)，如 Example 1-65 所示，当对象为一个容器时，所有的成员都被递归地复制了。

1.12.0.2. Example 1-65. 使用 copy 模块复制集合(Collections)

```
File: copy-example-2.py



import copy


```

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

b = copy.deepcopy(a)



print "before", "=>"

print a

print b



# modify original

a[0][0] = 0

a[1] = None



print "after", "=>"

print a

print b
```

```
before =>

[[1], [2], [3]]
```

```
[[1], [2], [3]]

after =>

[[0], None, [3]]

[[1], [2], [3]]
```

1.13. sys 模块

sys 模块提供了许多函数和变量来处理 Python 运行时环境的不同部分.

1.13.1. 处理命令行参数

在解释器启动后, argv 列表包含了传递给脚本的所有参数, 如 Example 1-66 所示. 列表的第一个元素为脚本自身的名称.

1.13.1.1. Example 1-66. 使用 sys 模块获得脚本的参数

```
File: sys-argv-example-1.py



import sys
```

```
print "script name is", sys.argv[0]
```

```
```

```
if len(sys.argv) > 1:

    print "there are", len(sys.argv)-1,
"arguments:"

    for arg in sys.argv[1:]:

        print arg

else:

    print "there are no arguments!"



script name is sys-argv-example-1.py

there are no arguments!
```

如果是从标准输入读入脚本（比如 "python < sys-argv-example-1.py"），脚本的名称将被设置为空串. 如果把脚本作为字符串传递给 python（使用 -c 选项），脚本名会被设置为 "-c".

1.13.2. 处理模块

path 列表是一个由目录名构成的列表，Python 从中查找扩展模块（ Python 源模块，编译模块,或者二进制扩展）. 启动 Python 时,这个列表从根据内建规则，PYTHONPATH 环境变量的内容，以及注册表（ Windows 系统)等进行初始化. 由于它只是一个普通的列表，你可以在程序中对它进行操作，如 Example 1-67 所示.

1.13.2.1. Example 1-67. 使用 sys 模块操作模块搜索路径

```
File: sys-path-example-1.py
```

```
import sys

print "path has", len(sys.path), "members"

# add the sample directory to the path
sys.path.insert(0, "samples")
import sample

# nuke the path
sys.path = []
import random # oops!
```

```
path has 7 members
this is the sample module!
```

```
Traceback (innermost last):
  File "sys-path-example-1.py", line 11, in ?
```

```
    import random # oops!

ImportError: No module named random
```

builtin_module_names 列表包含 Python 解释器中所有内建模块的名称, Example 1-68 给出了它的样例代码.

1.13.2.2. Example 1-68. 使用 sys 模块查找内建模块

```
File: sys-builtin-module-names-example-1.py



import sys



def dump(module):

    print module, "=>",

    if module in sys.builtin_module_names:

        print "<BUILTIN>"

    else:

        module = __import__(module)

        print module.__file__
```

```
dump("os")

dump("sys")

dump("string")

dump("strop")

dump("zlib")



os => C:\python\lib\os.pyc

sys => <BUILTIN>

string => C:\python\lib\string.pyc

strop => <BUILTIN>

zlib => C:\python\zlib.pyd
```

modules 字典包含所有加载的模块. import 语句在从磁盘导入内容之前会先检查这个字典.

正如你在 Example 1-69 中所见到的, Python 在处理你的脚本之前就已经导入了很多模块.

1.13.2.3. Example 1-69. 使用 sys 模块查找已导入的模块

```
File: sys-modules-example-1.py



import sys
```

```
print sys.modules.keys()
```

```
['os.path', 'os', 'exceptions', '__main__',
'ntpath', 'strop', 'nt',

'sys', '__builtin__', 'site', 'signal',
'UserDict', 'string', 'stat']
```

1.13.3. 处理引用记数

getrefcount 函数（如 Example 1-70 所示）返回给定对象的引用记数 - 也就是这个对象使用次数. Python 会跟踪这个值, 当它减少为 0 的时候, 就销毁这个对象.

1.13.3.1. Example 1-70. 使用 sys 模块获得引用记数

```
File: sys-getrefcount-example-1.py

import sys
```

```
variable = 1234
```

```
print sys.getrefcount(0)
```

```
print sys.getrefcount(variable)

print sys.getrefcount(None)



50

3

192
```

注意这个值总是比实际的数量大，因为该函数本身在确定这个值的时候依赖这个对象.

== 检查主机平台===

Example 1-71 展示了 platform 变量，它包含主机平台的名称.

1.13.3.2. Example 1-71. 使用 sys 模块获得当前平台

```
File: sys-platform-example-1.py



import sys
```

```
#
```

```
# emulate "import os.path" (sort of)...
```

```
if sys.platform == "win32":

    import ntpath

    pathmodule = ntpath

elif sys.platform == "mac":

    import macpath

    pathmodule = macpath

else:

    # assume it's a posix platform

    import posixpath

    pathmodule = posixpath



print pathmodule
```

典型的平台有 Windows 9X/NT(显示为 `win32`), 以及 Macintosh(显示为 `mac`). 对于 Unix 系统而言, platform 通常来自 `"uname -r"` 命令的输出, 例如 `irix6` , `linux2` , 或者 `sunos5` (Solaris).

1.13.4. 跟踪程序

`setprofiler` 函数允许你配置一个分析函数(profiling function). 这个函数会在每次调用某个函数或方法时被调用(明确或隐含的), 或是遇到异常的时候被调用. 让我们看看 Example 1-72 的代码.

1.13.4.1. Example 1-72. 使用 sys 模块配置分析函数

```
File: sys-setprofiler-example-1.py



import sys



def test(n):

    j = 0

    for i in range(n):

        j = j + i

    return n



def profiler(frame, event, arg):

    print event, frame.f_code.co_name,
frame.f_lineno, "->", arg



# profiler is activated on the next call, return, or
exception

# 分析函数将在下次函数调用, 返回, 或异常时激活

sys.setprofile(profiler)
```

```
# profile this function call
# 分析这次函数调用
test(1)


# disable profiler
# 禁用分析函数
sys.setprofile(None)


# don't profile this call
# 不会分析这次函数调用
test(2)
```

```
call test 3 -> None
return test 7 -> 1
```

基于该函数, profile 模块提供了一个完整的分析器框架.

Example 1-73 中的 settrace 函数与此类似, 但是 trace 函数会在解释器每执行到新的一行时被调用.

1.13.4.2. Example 1-73. 使用 sys 模块配置单步跟踪函数

File: sys-settrace-example-1.py

```
import sys



def test(n):
    j = 0
    for i in range(n):
        j = j + i
    return n
```

```
def tracer(frame, event, arg):
```

```
    print event, frame.f_code.co_name,
frame.f_lineno, "->", arg

    return tracer



# tracer is activated on the next call, return, or
exception
```

```
# 跟踪器将在下次函数调用，返回，或异常时激活
sys.settrace(tracer)


# trace this function call
# 跟踪这次函数调用
test(1)


# disable tracing
# 禁用跟踪器
sys.settrace(None)


# don't trace this call
# 不会跟踪这次函数调用
test(2)


call test 3 -> None
line test 3 -> None
```

```
line test 4 -> None

line test 5 -> None

line test 5 -> None

line test 6 -> None

line test 5 -> None

line test 7 -> None

return test 7 -> 1
```

基于该函数提供的跟踪功能, pdb 模块提供了完整的调试(debug)框架.

1.13.5. 处理标准输出/输入

stdin, stdout, 以及 stderr 变量包含与标准 I/O 流对应的流对象. 如果需要更好地控制输出,而 print 不能满足你的要求, 它们就是你所需要的. 你也可以 *替换* 它们, 这时候你就可以重定向输出和输入到其它设备(device), 或者以非标准的方式处理它们. 如 Example 1-74 所示.

1.13.5.1. Example 1-74. 使用 sys 重定向输出

```
File: sys-stdout-example-1.py



import sys

import string


```

```
class Redirect:



    def _ _init_ _(self, stdout):

        self.stdout = stdout



    def write(self, s):

        self.stdout.write(string.lower(s))



# redirect standard output (including the print
statement)

# 重定向标准输出(包括 print 语句)

old_stdout = sys.stdout
```

```
sys.stdout = Redirect(sys.stdout)
```

```


print "HEJA SVERIGE",

print "FRISKT HUM\303\226R"


```

```
# restore standard output

# 恢复标准输出

sys.stdout = old_stdout



print "M\303\205\303\205\303\205\303\205L!"



heja sverige friskt hum\303\266r

M\303\205\303\205\303\205\303\205L!
```

要重定向输出只要创建一个对象，并实现它的 `write` 方法.

(除非 C 类型的实例外:Python 使用一个叫做 `softspace` 的整数属性来控制输出中的空白. 如果没有这个属性，Python 将把这个属性附加到这个对象上. 你不需要在使用 Python 对象时担心，但是在重定向到一个 C 类型时，你应该确保该类型支持 `softspace` 属性.)

1.13.6. 退出程序

执行至主程序的末尾时,解释器会自动退出. 但是如果需要中途退出程序，你可以调用 `sys.exit` 函数，它带有一个可选的整数参数返回给调用它的程序. Example 1-75 给出了范例.

1.13.6.1. Example 1-75. 使用 sys 模块退出程序

```
File: sys-exit-example-1.py



import sys
```

```
print "hello"

sys.exit(1)

print "there"
```

```
hello
```

注意 sys.exit 并不是立即退出. 而是引发一个 *SystemExit* 异常. 这意味着你可以在主程序中捕获对 sys.exit 的调用, 如 Example 1-76 所示.

1.13.6.2. Example 1-76. 捕获 sys.exit 调用

```
File: sys-exit-example-2.py

import sys

print "hello"

try:
```

```
    sys.exit(1)

except SystemExit:

    pass



print "there"



hello

there
```

如果准备在退出前自己清理一些东西(比如删除临时文件), 你可以配置一个 "退出处理函数"(exit handler), 它将在程序退出的时候自动被调用. 如 Example 1-77 所示.

1.13.6.3. Example 1-77. 另一种捕获 sys.exit 调用的方法

```
File: sys-exitfunc-example-1.py



import sys



def exitfunc():

    print "world"
```

```
sys.exitfunc = exitfunc



print "hello"

sys.exit(1)

print "there" # never printed # 不会被 print



hello

world
```

在 Python 2.0 以后，你可以使用 `atexit` 模块来注册多个退出处理函数.

1.14. atexit 模块

(用于 2.0 版本及以上) `atexit` 模块允许你注册一个或多个终止函数(暂且这么叫)，这些函数将在解释器终止前被自动调用.

调用 `register` 函数，便可以将函数注册为终止函数，如 Example 1-78 所示. 你也可以添加更多的参数，这些将作为 `exit` 函数的参数传递.

1.14.0.1. Example 1-78. 使用 atexit 模块

```
File: atexit-example-1.py



```

```
import atexit



def exit(*args):

    print "exit", args



# register two exit handler

atexit.register(exit)

atexit.register(exit, 1)
```

```
atexit.register(exit, "hello", "world")
```

```
exit ('hello', 'world')

exit (1,)

exit ()
```

该模块其实是一个对 `sys.exitfunc` 钩子(hook)的简单封装.

1.15. time 模块

`time` 模块提供了一些处理日期和一天内时间的函数. 它是建立在 C 运行时库的简单封装.

给定的日期和时间可以被表示为浮点型(从参考时间, 通常是 1970.1.1 到现在经过的秒数. 即 Unix 格式), 或者一个表示时间的 struct (类元组).

1.15.1. 获得当前时间

Example 1-79 展示了如何使用 time 模块获取当前时间.

1.15.1.1. Example 1-79. 使用 time 模块获取当前时间

```
File: time-example-1.py



import time
```

```

```

```
now = time.time()



print now, "seconds since", time.gmtime(0)[:6]

print

print "or in other words:"

print "- local time:", time.localtime(now)

print "- utc:", time.gmtime(now)
```

```
937758359.77 seconds since (1970, 1, 1, 0, 0, 0)
```

```
or in other words:

- local time: (1999, 9, 19, 18, 25, 59, 6, 262, 1)

- utc: (1999, 9, 19, 16, 25, 59, 6, 262, 0)
```

localtime 和 gmtime 返回的类元组包括年, 月, 日, 时, 分, 秒, 星期, 一年的第几天, 日光标志. 其中年是一个四位数(在有千年虫问题的平台上另有规定, 但还是四位数), 星期从星期一(数字 0 代表)开始, 1 月 1 日是一年的第一天.

1.15.2. 将时间值转换为字符串

你可以使用标准的格式化字符串把时间对象转换为字符串, 不过 time 模块已经提供了许多标准转换函数, 如 Example 1-80 所示.

1.15.2.1. Example 1-80. 使用 time 模块格式化时间输出

```
File: time-example-2.py



import time



now = time.localtime(time.time())



print time.asctime(now)

print time.strftime("%y/%m/%d %H:%M", now)
```

```
print time.strftime("%a %b %d", now)

print time.strftime("%c", now)

print time.strftime("%I %p", now)

print time.strftime("%Y-%m-%d %H:%M:%S %Z", now)



# do it by hand...
```

```
year, month, day, hour, minute, second, weekday,
yearday, daylight = now
```

```
print "%04d-%02d-%02d" % (year, month, day)

print "%02d:%02d:%02d" % (hour, minute, second)

print ("MON", "TUE", "WED", "THU", "FRI", "SAT",
"SUN")[weekday], yearday
```

```
Sun Oct 10 21:39:24 1999

99/10/10 21:39

Sun Oct 10

Sun Oct 10 21:39:24 1999

09 PM
```

```
1999-10-10 21:39:24 CEST

1999-10-10

21:39:24

SUN 283
```

1.15.3. 将字符串转换为时间对象

在一些平台上，`time` 模块包含了 `strptime` 函数，它的作用与 `strftime` 相反. 给定一个字符串和模式，它返回相应的时间对象，如 Example 1-81 所示.

1.15.3.1. Example 1-81. 使用 time.strptime 函数解析时间

```
File: time-example-6.py



import time



# make sure we have a strptime function!

# 确认有函数 strptime

try:

    strptime = time.strptime

except AttributeError:
```

```
    from strptime import strptime



print strptime("31 Nov 00", "%d %b %y")

print strptime("1 Jan 70 1:30pm", "%d %b %y %I:%M%p")
```

只有在系统的 C 库提供了相应的函数的时候, `time.strptime` 函数才可以使用. 对于没有提供标准实现的平台, Example 1-82 提供了一个不完全的实现.

1.15.3.2. Example 1-82. strptime 实现

```
File: strptime.py



import re

import string



MONTHS = ["Jan", "Feb", "Mar", "Apr", "May", "Jun",
"Jul", "Aug",

          "Sep", "Oct", "Nov", "Dec"]



SPEC = {
```

```
    # map formatting code to a regular expression
fragment

    "%a": "(?P<weekday>[a-z]+)",

    "%A": "(?P<weekday>[a-z]+)",

    "%b": "(?P<month>[a-z]+)",

    "%B": "(?P<month>[a-z]+)",
```

```
    "%C": "(?P<century>\d\d?)",
```

```
    "%d": "(?P<day>\d\d?)",

    "%D":
"(?P<month>\d\d?)/(?P<day>\d\d?)/(?P<year>\d\d)",

    "%e": "(?P<day>\d\d?)",

    "%h": "(?P<month>[a-z]+)",

    "%H": "(?P<hour>\d\d?)",

    "%I": "(?P<hour12>\d\d?)",

    "%j": "(?P<yearday>\d\d?\d?)",

    "%m": "(?P<month>\d\d?)",

    "%M": "(?P<minute>\d\d?)",

    "%p": "(?P<ampm12>am|pm)",
```

```
    "%R": "(?P<hour>\d\d?):(?P<minute>\d\d?)",

    "%S": "(?P<second>\d\d?)",

    "%T":
"(?P<hour>\d\d?):(?P<minute>\d\d?):(?P<second>\d\
d?)",

    "%U": "(?P<week>\d\d)",

    "%w": "(?P<weekday>\d)",

    "%W": "(?P<weekday>\d\d)",

    "%y": "(?P<year>\d\d)",

    "%Y": "(?P<year>\d\d\d\d)",

    "%%": "%"

}



class TimeParser:

    def _ _init_ _(self, format):

        # convert strptime format string to regular
expression

        format =
string.join(re.split("(?:\s|%t|%n)+", format))
```

```
        pattern = []

        try:

            for spec in re.findall("%\w|%%|.",
format):

                if spec[0] == "%":

                    spec = SPEC[spec]

                pattern.append(spec)
```

```
        except KeyError:
```

```
            raise ValueError, "unknown
specificer: %s" % spec

        self.pattern = re.compile("(?i)" +
string.join(pattern, ""))

    def match(self, daytime):

        # match time string

        match = self.pattern.match(daytime)

        if not match:

            raise ValueError, "format mismatch"

        get = match.groupdict().get

        tm = [0] * 9
```

```
            # extract date elements
            y = get("year")
            if y:
                y = int(y)
                if y < 68:
                    y = 2000 + y
                elif y < 100:
                    y = 1900 + y
                tm[0] = y
            m = get("month")
            if m:
                if m in MONTHS:
                    m = MONTHS.index(m) + 1
                tm[1] = int(m)
            d = get("day")
            if d: tm[2] = int(d)
            # extract time elements
```

```
            h = get("hour")

            if h:

                tm[3] = int(h)

            else:

                h = get("hour12")

                if h:

                    h = int(h)
```

```
                    if string.lower(get("ampm12", ""))
== "pm":
```

```
                        h = h + 12

                    tm[3] = h

            m = get("minute")

            if m: tm[4] = int(m)

            s = get("second")

            if s: tm[5] = int(s)

            # ignore weekday/yearday for now

            return tuple(tm)
```

```
def strptime(string, format="%a %b %d %H:%M:%S %Y"):
    return TimeParser(format).match(string)

if _ _name_ _ == "_ _main_ _":
    # try it out
    import time
    print strptime("2000-12-20 01:02:03",
"%Y-%m-%d %H:%M:%S")
    print strptime(time.ctime(time.time()))
```

```
(2000, 12, 20, 1, 2, 3, 0, 0, 0)
(2000, 11, 15, 12, 30, 45, 0, 0, 0)
```

1.15.4. 转换时间值

将时间元组转换回时间值非常简单, 至少我们谈论的当地时间 (local time) 如此. 只要把时间元组传递给 mktime 函数, 如 Example 1-83 所示.

1.15.4.1. Example 1-83. 使用 time 模块将本地时间元组转换为时间值(整数)

```
File: time-example-3.py



import time



t0 = time.time()

tm = time.localtime(t0)



print tm
```

```
print t0
```

```
print time.mktime(tm)



(1999, 9, 9, 0, 11, 8, 3, 252, 1)

936828668.16

936828668.0
```

但是, 1.5.2 版本的标准库没有提供能将 UTC 时间 (Universal Time, Coordinated: 特林威治标准时间)转换为时间值的函数 (Python 和对应底层

C 库都没有提供). Example 1-84 提供了该函数的一个 Python 实现, 称为 timegm .

1.15.4.2. Example 1-84. 将 UTC 时间元组转换为时间值(整数)

```
File: time-example-4.py



import time



def _d(y, m, d,
days=(0,31,59,90,120,151,181,212,243,273,304,334,
365)):

    # map a date to the number of days from a
reference point

    return (((y - 1901)*1461)/4 + days[m-1] + d +

        ((m > 2 and not y % 4 and (y % 100 or not y %
400)) and 1))



def timegm(tm, epoch=_d(1970,1,1)):

    year, month, day, h, m, s = tm[:6]

    assert year >= 1970

    assert 1 <= month <= 12
```

```
    return (_d(year, month, day) - epoch)*86400 +
h*3600 + m*60 + s




t0 = time.time()

tm = time.gmtime(t0)




print tm




print t0

print timegm(tm)
```

```
(1999, 9, 8, 22, 12, 12, 2, 251, 0)
```

```
936828732.48

936828732
```

从 1.6 版本开始, `calendar` 模块提供了一个类似的函数 `calendar.timegm` .

1.15.5. Timing 相关

time 模块可以计算 Python 程序的执行时间，如 Example 1-85 所示. 你可以测量 "wall time"(real world time)，或是"进程时间"(消耗的 CPU 时间).

1.15.5.1. Example 1-85. 使用 time 模块评价算法

```
File: time-example-5.py



import time



def procedure():

    time.sleep(2.5)



# measure process time

t0 = time.clock()

procedure()

print time.clock() - t0, "seconds process time"



# measure wall time

t0 = time.time()

procedure()
```

```
print time.time() - t0, "seconds wall time"



0.0 seconds process time

2.50903499126 seconds wall time
```

并不是所有的系统都能测量真实的进程时间. 一些系统中(包括 Windows), `clock` 函数通常测量从程序启动到测量时的 wall time.

进程时间的精度受限制. 在一些系统中, 它超过 30 分钟后进程会被清理. (原文: On many systems, it wraps around after just over 30 minutes.)

另参见 `timing` 模块(Windows 下的朋友不用忙活了,没有地~), 它可以测量两个事件之间的 wall time.

1.16. types 模块

`types` 模块包含了标准解释器定义的所有类型的类型对象, 如 Example 1-86 所示. 同一类型的所有对象共享一个类型对象. 你可以使用 `is` 来检查一个对象是不是属于某个给定类型.

1.16.0.1. Example 1-86. 使用 types 模块

```
File: types-example-1.py



import types



def check(object):

    print object,
```

```
    if type(object) is types.IntType:
        print "INTEGER",
    if type(object) is types.FloatType:
        print "FLOAT",
    if type(object) is types.StringType:
        print "STRING",
    if type(object) is types.ClassType:
        print "CLASS",
    if type(object) is types.InstanceType:
        print "INSTANCE",
    print

check(0)
check(0.0)
check("0")
```

```
class A:
    pass


class B:
    pass


check(A)
check(B)
```

```
a = A()
b = B()


check(a)
check(b)


0 INTEGER
0.0 FLOAT
```

```
0 STRING

A CLASS

B CLASS

<A instance at 796960> INSTANCE

<B instance at 796990> INSTANCE
```

注意所有的类都具有相同的类型，所有的实例也是一样. 要测试一个类或者实例所属的类，可以使用内建的 `issubclass` 和 `isinstance` 函数.

`types` 模块在第一次引入的时候会破坏当前的异常状态. 也就是说，不要在异常处理语句块中导入该模块（*或其他会导入它的模块*）.

1.17. gc 模块

（可选，2.0 及以后版本）`gc` 模块提供了到内建循环垃圾收集器的接口.

Python 使用引用记数来跟踪什么时候销毁一个对象；一个对象的最后一个引用一旦消失，这个对象就会被销毁.

从 2.0 版开始，Python 还提供了一个循环垃圾收集器，它每隔一段时间执行. 这个收集器查找指向自身的数据结构，并尝试破坏循环. 如 Example 1-87 所示.

你可以使用 `gc.collect` 函数来强制完整收集. 这个函数将返回收集器销毁的对象的数量.

1.17.0.1. Example 1-87. 使用 gc 模块收集循环引用垃圾

```
File: gc-example-1.py



import gc
```

```
# create a simple object that links to itself

class Node:



    def _ _init_ _(self, name):

        self.name = name

        self.parent = None

        self.children = []
```

```

```

```
    def addchild(self, node):

        node.parent = self

        self.children.append(node)



    def _ _repr_ _(self):

        return "<Node %s at %x>" % (repr(self.name),
id(self))
```

```
# set up a self-referencing structure

root = Node("monty")



root.addchild(Node("eric"))

root.addchild(Node("john"))

root.addchild(Node("michael"))



# remove our only reference

del root
```

```

```

```
print gc.collect(), "unreachable objects"

print gc.collect(), "unreachable objects"



12 unreachable objects

0 unreachable objects
```

如果你确定你的程序不会创建自引用的数据结构, 你可以使用 gc.disable 函数禁用垃圾收集, 调用这个函数以后, Python 的工作方式将与 1.5.2 或更早的版本相同.

2. 更多标准模块

"Now, imagine that your friend kept complaining that she didn't want to visit you since she found it too hard to climb up the drain pipe, and you kept telling her to use the friggin' stairs like everyone else..."
- eff-bot, June 1998

2.1. 概览

本章叙述了许多在 Python 程序中广泛使用的模块. 当然, 在大型的 Python 程序中不使用这些模块也是可以的, 但如果使用会节省你不少时间.

2.1.1. 文件与流

`fileinput` 模块可以让你更简单地向不同的文件写入内容. 该模块提供了一个简单的封装类, 一个简单的 `for-in` 语句就可以循环得到一个或多个文本文件的内容.

`StringIO` 模块 (以及 `cStringIO` 模块, 作为一个的变种) 实现了一个工作在内存的文件对象. 你可以在很多地方用 `StringIO` 对象替换普通的文件对象.

2.1.2. 类型封装

`UserDict`, `UserList`, 以及 `UserString` 是对应内建类型的顶层简单封装. 和内建类型不同的是, 这些封装是可以被继承的. 这在你需要一个和内建类型行为相似但由额外新方法的类的时候很有用.

2.1.3. 随机数字

`random` 模块提供了一些不同的随机数字生成器. `whrandom` 模块与此相似, 但允许你创建多个生成器对象.

> [!Feather 注: whrandom 在版本 2.1 时声明不支持.
> 请使用 random 替代.]

2.1.4. 加密算法

md5 和 sha 模块用于计算密写的信息标记(cryptographically strong message signatures , 所谓的 "message digests", 信息摘要).

crypt 模块实现了 DES 样式的单向加密. 该模块只在 Unix 系统下可用.

rotor 模块提供了简单的双向加密. 版本 2.4 以后的朋友可以不用忙活了.

> [!Feather 注: 它在版本 2.3 时申明不支持, 因为它的加密运算不安全.]

2.2. fileinput 模块

fileinput 模块允许你循环一个或多个文本文件的内容, 如 Example 2-1 所示.

2.2.0.1. Example 2-1. 使用 fileinput 模块循环一个文本文件

File: fileinput-example-1.py

```
import fileinput
import sys


for line in fileinput.input("samples/sample.txt"):
    sys.stdout.write("-> ")
    sys.stdout.write(line)
```

```
-> We will perhaps eventually be writing only small

-> modules which are identified by name as they are

-> used to build larger ones, so that devices like

-> indentation, rather than delimiters, might become

-> feasible for expressing local structure in the

-> source language.

->      -- Donald E. Knuth, December 1974
```

你也可以使用 `fileinput` 模块获得当前行的元信息 (meta information). 其中包括 `isfirstline`, `filename`, `lineno`, 如 Example 2-2 所示.

2.2.0.2. Example 2-2. 使用 fileinput 模块处理多个文本文件

```
File: fileinput-example-2.py



import fileinput

import glob

import string, sys
```

```
for line in
fileinput.input(glob.glob("samples/*.txt")):

    if fileinput.isfirstline(): # first in a file?
```

```
        sys.stderr.write("-- reading %s --\n" %
fileinput.filename())
```

```
    sys.stdout.write(str(fileinput.lineno()) + " "
+ string.upper(line))
```

```
-- reading samples\sample.txt --

1 WE WILL PERHAPS EVENTUALLY BE WRITING ONLY SMALL

2 MODULES WHICH ARE IDENTIFIED BY NAME AS THEY ARE

3 USED TO BUILD LARGER ONES, SO THAT DEVICES LIKE

4 INDENTATION, RATHER THAN DELIMITERS, MIGHT BECOME

5 FEASIBLE FOR EXPRESSING LOCAL STRUCTURE IN THE

6 SOURCE LANGUAGE.

7    -- DONALD E. KNUTH, DECEMBER 1974
```

文本文件的替换操作很简单. 只需要把 inplace 关键字参数设置为 1 , 传递给 input 函数, 该模块会帮你做好一切. Example 2-3 展示了这些.

2.2.0.3. Example 2-3. 使用 fileinput 模块将 CRLF 改为 LF

```
File: fileinput-example-3.py
```

```
import fileinput, sys
```

```
for line in fileinput.input(inplace=1):
    # convert Windows/DOS text files to Unix files
    if line[-2:] == "\r\n":
        line = line[:-2] + "\n"
    sys.stdout.write(line)
```

2.3. shutil 模块

shutil 实用模块包含了一些用于复制文件和文件夹的函数. Example 2-4 中使用的 copy 函数使用和 Unix 下 cp 命令基本相同的方式复制一个文件.

2.3.0.1. Example 2-4. 使用 shutil 复制文件

```
File: shutil-example-1.py
```

```
import shutil
```

```
import os
```

```
for file in os.listdir("."):
```

```
    if os.path.splitext(file)[1] == ".py":

        print file

        shutil.copy(file, os.path.join("backup",
file))
```

```
aifc-example-1.py

anydbm-example-1.py

array-example-1.py

...
```

copytree 函数用于复制整个目录树 (与 cp -r 相同), 而 rmtree 函数用于删除整个目录树 (与 rm -r). 如 Example 2-5 所示.

2.3.0.2. Example 2-5. 使用 shutil 模块复制/删除目录树

```
File: shutil-example-2.py
```

```
import shutil
```

```
import os
```

```

```

```
SOURCE = "samples"

BACKUP = "samples-bak"



# create a backup directory

shutil.copytree(SOURCE, BACKUP)



print os.listdir(BACKUP)



# remove it

shutil.rmtree(BACKUP)



print os.listdir(BACKUP)



['sample.wav', 'sample.jpg', 'sample.au',
'sample.msg', 'sample.tgz',
```

```
...

Traceback (most recent call last):

 File "shutil-example-2.py", line 17, in ?

   print os.listdir(BACKUP)

os.error: No such file or directory
```

2.4. tempfile 模块

Example 2-6 中展示的 tempfile 模块允许你快速地创建名称唯一的临时文件供使用.

2.4.0.1. Example 2-6. 使用 tempfile 模块创建临时文件

```
File: tempfile-example-1.py



import tempfile

import os



tempfile = tempfile.mktemp()
```

```
print "tempfile", "=>", tempfile
```

```
file = open(tempfile, "w+b")
```

```
file.write("*" * 1000)

file.seek(0)

print len(file.read()), "bytes"

file.close()



try:

    # must remove file when done

    os.remove(tempfile)

except OSError:

    pass



tempfile => C:\TEMP\~160-1

1000 bytes
```

TemporaryFile 函数会自动挑选合适的文件名，并打开文件，如 Example 2-7 所示. 而且它会确保该文件在关闭的时候会被删除.（在 Unix 下，你可以删除一个已打开的文件，这时文件关闭时它会被自动删除. 在其他平台上，这通过一个特殊的封装类实现.）

2.4.0.2. Example 2-7. 使用 tempfile 模块打开临时文件

```
File: tempfile-example-2.py



import tempfile



file = tempfile.TemporaryFile()



for i in range(100):

    file.write("*" * 100)



file.close() # removes the file!
```

2.5. StringIO 模块

Example 2-8 展示了 StringIO 模块的使用. 它实现了一个工作在内存的文件对象（内存文件）. 在大多需要标准文件对象的地方都可以使用它来替换.

2.5.0.1. Example 2-8. 使用 StringIO 模块从内存文件读入内容

```
File: stringio-example-1.py
```

```
import StringIO
```

```
MESSAGE = "That man is depriving a village somewhere
of a computer scientist."

file = StringIO.StringIO(MESSAGE)

print file.read()
```

```
That man is depriving a village somewhere of a
computer scientist.
```

StringIO 类实现了内建文件对象的所有方法, 此外还有 getvalue 方法用来返回它内部的字符串值. Example 2-9 展示了这个方法.

2.5.0.2. Example 2-9. 使用 StringIO 模块向内存文件写入内容

```
File: stringio-example-2.py
```

```
import StringIO
```

```
file = StringIO.StringIO()
```

```
file.write("This man is no ordinary man. ")

file.write("This is Mr. F. G. Superman.")



print file.getvalue()



This man is no ordinary man. This is Mr. F. G.
Superman.
```

StringIO 可以用于重新定向 Python 解释器的输出，如 Example 2-10 所示.

2.5.0.3. Example 2-10. 使用 StringIO 模块捕获输出

```
File: stringio-example-3.py



import StringIO

import string, sys
```

```
stdout = sys.stdout
```

```
sys.stdout = file = StringIO.StringIO()
```

```
print """
According to Gbaya folktales, trickery and guile
are the best ways to defeat the python, king of
snakes, which was hatched from a dragon at the
world's start. -- National Geographic, May 1997
"""

sys.stdout = stdout

print string.upper(file.getvalue())

ACCORDING TO GBAYA FOLKTALES, TRICKERY AND GUILE
ARE THE BEST WAYS TO DEFEAT THE PYTHON, KING OF
```

```
SNAKES, WHICH WAS HATCHED FROM A DRAGON AT THE

WORLD'S START. -- NATIONAL GEOGRAPHIC, MAY 1997
```

2.6. cStringIO 模块

cStringIO 是一个可选的模块，是 StringIO 的更快速实现. 它的工作方式和 StringIO 基本相同，但是它不可以被继承. Example 2-11 展示了 cStringIO 的用法，另参考前一节.

2.6.0.1. Example 2-11. 使用 cStringIO 模块

```
File: cstringio-example-1.py



import cStringIO



MESSAGE = "That man is depriving a village somewhere
of a computer scientist."



file = cStringIO.StringIO(MESSAGE)



print file.read()
```

```
That man is depriving a village somewhere of a
computer scientist.
```

为了让你的代码尽可能快，但同时保证兼容低版本的 Python ，你可以使用一个小技巧在 `cStringIO` 不可用时启用 `StringIO` 模块，如 Example 2-12 所示.

2.6.0.2. Example 2-12. 后退至 StringIO

```
File: cstringio-example-2.py



try:

    import cStringIO

    StringIO = cStringIO

except ImportError:

    import StringIO



print StringIO



<module 'StringIO' (built-in)>
```

2.7. mmap 模块

(2.0 新增) mmap 模块提供了操作系统内存映射函数的接口, 如 Example 2-13 所示. 映射区域的行为和字符串对象类似, 但数据是直接从文件读取的.

2.7.0.1. Example 2-13. 使用 mmap 模块

```
File: mmap-example-1.py



import mmap
```

```
import os



filename = "samples/sample.txt"



file = open(filename, "r+")

size = os.path.getsize(filename)



data = mmap.mmap(file.fileno(), size)



# basics
```

```
print data
```

```
print len(data), size
```

```
# use slicing to read from the file

# 使用切片操作读取文件

print repr(data[:10]), repr(data[:10])
```

```
# or use the standard file interface

# 或使用标准的文件接口

print repr(data.read(10)), repr(data.read(10))
```

```
<mmap object at 008A2A10>

302 302

'We will pe' 'We will pe'

'We will pe' 'rhaps even'
```

在 Windows 下，这个文件必须以既可读又可写的模式打开（`r+`，`w+`，或`a+`），否则 mmap 调用会失败.

> [!Feather 注：经本人测试，a+ 模式是完全可以的，原文只有 r+ 和 w+]

Example 2-14 展示了内存映射区域的使用，在很多地方它都可以替换普通字符串使用，包括正则表达式和其他字符串操作.

2.7.0.2. Example 2-14. 对映射区域使用字符串方法和正则表达式

```
File: mmap-example-2.py



import mmap

import os, string, re



def mapfile(filename):

    file = open(filename, "r+")

    size = os.path.getsize(filename)

    return mmap.mmap(file.fileno(), size)



data = mapfile("samples/sample.txt")
```

```
# search

index = data.find("small")

print index, repr(data[index-5:index+15])



# regular expressions work too!

m = re.search("small", data)

print m.start(), m.group()



43 'only small\015\012modules '

43 small
```

2.8. UserDict 模块

UserDict 模块包含了一个可继承的字典类 (事实上是对内建字典类型的 Python 封装).

Example 2-15 展示了一个增强的字典类, 允许对字典使用 "加/+" 操作并提供了接受关键字参数的构造函数.

2.8.0.1. Example 2-15. 使用 UserDict 模块

```
File: userdict-example-1.py
```

```
import UserDict
```

```
class FancyDict(UserDict.UserDict):

    def _ _init_ _(self, data = {}, **kw):
        UserDict.UserDict._ _init_ _(self)
        self.update(data)
        self.update(kw)

    def _ _add_ _(self, other):
        dict = FancyDict(self.data)
        dict.update(b)
        return dict

a = FancyDict(a = 1)
b = FancyDict(b = 2)
```

```

print a + b
```

```
{'b': 2, 'a': 1}
```

2.9. UserList 模块

UserList 模块包含了一个可继承的列表类（事实上是对内建列表类型的 Python 封装).

在 Example 2-16 中, *AutoList* 实例类似一个普通的列表对象, 但它允许你通过赋值为列表添加项目.

2.9.0.1. Example 2-16. 使用 UserList 模块

```
File: userlist-example-1.py



import UserList



class AutoList(UserList.UserList):



    def __setitem__(self, i, item):
```

```
            if i == len(self.data):
```

```
                self.data.append(item)
```

```
            else:
                self.data[i] = item



list = AutoList()



for i in range(10):
    list[i] = i



print list



[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.10. UserString 模块

(2.0 新增) `UserString` 模块包含两个类, *UserString* 和 *MutableString*. 前者是对标准字符串类型的封装, 后者是一个变种, 允许你修改特定位置的字符(联想下列表就知道了).

注意 *MutableString* 并不是效率很好，许多操作是通过切片和字符串连接实现的. 如果性能很对你的脚本来说重要的话，你最好使用字符串片断的列表或者 `array` 模块. Example 2-17 展示了 `UserString` 模块.

2.10.0.1. Example 2-17. 使用 UserString 模块

```
File: userstring-example-1.py



import UserString



class MyString(UserString.MutableString):



    def append(self, s):

        self.data = self.data + s



    def insert(self, index, s):

        self.data = self.data[index:] + s +
self.data[index:]



    def remove(self, s):

        self.data = self.data.replace(s, "")
```

```
file = open("samples/book.txt")

text = file.read()

file.close()



book = MyString(text)



for bird in ["gannet", "robin", "nuthatch"]:

    book.remove(bird)



print book
```

```
...

C: The one without the !

P: The one without the -!!! They've ALL got the !!
It's a

Standard British Bird, the , it's in all the books!!!

...
```

2.11. traceback 模块

Example 2-18 展示了 traceback 模块允许你在程序里打印异常的跟踪返回 (Traceback)信息, 类似未捕获异常时解释器所做的. 如 Example 2-18 所示.

2.11.0.1. Example 2-18. 使用 traceback 模块打印跟踪返回信息

```
File: traceback-example-1.py



# note! importing the traceback module messes up the

# exception state, so you better do that here and not

# in the exception handler

# 注意! 导入 traceback 会清理掉异常状态, 所以

# 最好别在异常处理代码中导入该模块

import traceback



try:

    raise SyntaxError, "example"

except:

    traceback.print_exc()
```

```
Traceback (innermost last):
  File "traceback-example-1.py", line 7, in ?
SyntaxError: example
```

Example 2-19 使用 StringIO 模块将跟踪返回信息放在字符串中.

2.11.0.2. Example 2-19. 使用 traceback 模块将跟踪返回信息复制到字符串

```
File: traceback-example-2.py

import traceback
import StringIO

try:
    raise IOError, "an i/o error occurred"
except:
    fp = StringIO.StringIO()
    traceback.print_exc(file=fp)
    message = fp.getvalue()
```

```
    print "failure! the error was:", repr(message)



failure! the error was: 'Traceback (innermost
last):\012  File

"traceback-example-2.py", line 5, in ?\012IOError:
an i/o error

occurred\012'
```

你可以使用 `extract_tb` 函数格式化跟踪返回信息, 得到包含错误信息的列表, 如 Example 2-20 所示.

2.11.0.3. Example 2-20. 使用 traceback Module 模块编码 Traceback 对象

```
File: traceback-example-3.py



import traceback

import sys



def function():

    raise IOError, "an i/o error occurred"
```

```
try:
    function()
except:
    info = sys.exc_info()
    for file, lineno, function, text in
traceback.extract_tb(info[2]):
        print file, "line", lineno, "in", function
        print "=>", repr(text)
    print "** %s: %s" % info[:2]
```

```
traceback-example-3.py line 8 in ?
=> 'function()'
traceback-example-3.py line 5 in function
=> 'raise IOError, "an i/o error occurred"'
** exceptions.IOError: an i/o error occurred
```

2.12. errno 模块

errno 模块定义了许多的符号错误码，比如 ENOENT ("没有该目录入口") 以及 EPERM ("权限被拒绝")．它还提供了一个映射到对应平台数字错误代码的字典．Example 2-21 展示了如何使用 errno 模块．

在大多情况下，*IOError* 异常会提供一个二元元组，包含对应数值错误代码和一个说明字符串．如果你需要区分不同的错误代码，那么最好在可能的地方使用符号名称．

2.12.0.1. Example 2-21. 使用 errno 模块

```
File: errno-example-1.py



import errno



try:

    fp = open("no.such.file")

except IOError, (error, message):

    if error == errno.ENOENT:

        print "no such file"

    elif error == errno.EPERM:

        print "permission denied"

    else:

        print message
```

```
no such file
```

Example 2-22 绕了些无用的弯子, 不过它很好地说明了如何使用 errorcode 字典把数字错误码映射到符号名称(symbolic name).

2.12.0.2. Example 2-22. 使用 errorcode 字典

```
File: errno-example-2.py

import errno

try:
    fp = open("no.such.file")
except IOError, (error, message):
    print error, repr(message)
    print errno.errorcode[error]

# 2 'No such file or directory'
# ENOENT
```

2.13. getopt 模块

getopt 模块包含用于抽出命令行选项和参数的函数, 它可以处理多种格式的选项. 如 Example 2-23 所示.

其中第 2 个参数指定了允许的可缩写的选项. 选项名后的冒号(:) 意味这这个选项必须有额外的参数.

2.13.0.1. Example 2-23. 使用 getopt 模块

```
File: getopt-example-1.py



import getopt

import sys



# simulate command-line invocation

# 模仿命令行参数

sys.argv = ["myscript.py", "-l", "-d", "directory",
"filename"]



# process options

# 处理选项
```

```
opts, args = getopt.getopt(sys.argv[1:], "ld:")



long = 0

directory = None



for o, v in opts:

    if o == "-l":

        long = 1

    elif o == "-d":

        directory = v



print "long", "=", long

print "directory", "=", directory

print "arguments", "=", args



long = 1
```

```
directory = directory
```

```
arguments = ['filename']
```

为了让 getopt 查找长的选项，如 Example 2-24 所示，传递一个描述选项的列表做为第 3 个参数. 如果一个选项名称以等号(=) 结尾，那么它必须有一个附加参数.

2.13.0.2. Example 2-24. 使用 getopt 模块处理长选项

```
File: getopt-example-2.py



import getopt

import sys



# simulate command-line invocation

# 模仿命令行参数

sys.argv = ["myscript.py", "--echo", "--printer",
"lp01", "message"]



opts, args = getopt.getopt(sys.argv[1:], "ep:",
["echo", "printer="])



# process options
```

```
# 处理选项

echo = 0

printer = None



for o, v in opts:

    if o in ("-e", "--echo"):

        echo = 1

    elif o in ("-p", "--printer"):

        printer = v



print "echo", "=", echo

print "printer", "=", printer

print "arguments", "=", args



echo = 1

printer = lp01

arguments = ['message']
```

```
[!Feather 注: 我不知道大家明白没, 可以自己试下:

myscript.py -e -p lp01 message

myscript.py --echo --printer=lp01 message

]
```

2.14. getpass 模块

getpass 模块提供了平台无关的在命令行下输入密码的方法. 如 Example 2-25 所示.

getpass(prompt) 会显示提示字符串, 关闭键盘的屏幕反馈, 然后读取密码. 如果提示参数省略, 那么它将打印出 "Password:".

getuser() 获得当前用户名, 如果可能的话.

2.14.0.1. Example 2-25. 使用 getpass 模块

```
File: getpass-example-1.py



import getpass



usr = getpass.getuser()
```

```
pwd = getpass.getpass("enter password for user %s:
" % usr)



print usr, pwd



enter password for user mulder:

mulder trustno1
```

2.15. glob 模块

glob 根据给定模式生成满足该模式的文件名列表，和 Unix shell 相同.

这里的模式和正则表达式类似，但更简单. 星号(*) 匹配零个或更多个字符，问号(?) 匹配单个字符. 你也可以使用方括号来指定字符范围，例如 [0-9] 代表一个数字. 其他所有字符都代表它们本身.

glob(pattern) 返回满足给定模式的所有文件的列表. Example 2-26 展示了它的用法.

2.15.0.1. Example 2-26. 使用 glob 模块

```
File: glob-example-1.py



import glob


```

```
for file in glob.glob("samples/*.jpg"):

    print file



samples/sample.jpg
```

注意这里的 glob 返回完整路径名，这点和 os.listdir 函数不同. glob 事实上使用了 fnmatch 模块来完成模式匹配.

2.16. fnmatch 模块

fnmatch 模块使用模式来匹配文件名. 如 Example 2-27 所示.

模式语法和 Unix shell 中所使用的相同. 星号(*) 匹配零个或更多个字符，问号(?) 匹配单个字符. 你也可以使用方括号来指定字符范围，例如 [0-9] 代表一个数字. 其他所有字符都匹配它们本身.

2.16.0.1. Example 2-27. 使用 fnmatch 模块匹配文件

```
File: fnmatch-example-1.py



import fnmatch
```

```
import os
```

```


for file in os.listdir("samples"):
```

```
    if fnmatch.fnmatch(file, "*.jpg"):

        print file



sample.jpg
```

Example 2-28 中的 `translate` 函数可以将一个文件匹配模式转换为正则表达式.

2.16.0.2. Example 2-28. 使用 fnmatch 模块将模式转换为正则表达式

```
File: fnmatch-example-2.py



import fnmatch

import os, re



pattern = fnmatch.translate("*.jpg")



for file in os.listdir("samples"):

    if re.match(pattern, file):

        print file
```

```
print "(pattern was %s)" % pattern
```

```
sample.jpg
(pattern was .*\.jpg$)
```

glob 和 find 模块在内部使用 fnmatch 模块来实现.

2.17. random 模块

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
- John von Neumann, 1951

random 模块包含许多随机数生成器.

基本随机数生成器(基于 Wichmann 和 Hill, 1982 的数学运算理论) 可以通过很多方法访问, 如 Example 2-29 所示.

2.17.0.1. Example 2-29. 使用 random 模块获得随机数字

```
File: random-example-1.py
```

```
import random
```

```
for i in range(5):
```

```
    # random float: 0.0 <= number < 1.0
    print random.random(),

    # random float: 10 <= number < 20
    print random.uniform(10, 20),

    # random integer: 100 <= number <= 1000
    print random.randint(100, 1000),

    # random integer: even numbers in 100 <= number
< 1000
    print random.randrange(100, 1000, 2)
```

```
0.946842713956 19.5910069381 709 172
```

```
0.573613195398 16.2758417025 407 120
```

```
0.363241598013 16.8079747714 916 580
```

```
0.602115173978 18.386796935 531 774

0.526767588533 18.0783794596 223 344
```

注意这里的 randint 函数可以返回上界，而其他函数总是返回小于上界的值. 所有函数都有可能返回下界值.

Example 2-30 展示了 choice 函数，它用来从一个序列里分拣出一个随机项目. 它可以用于列表，元组，以及其他序列(当然，非空的).

2.17.0.2. Example 2-30. 使用 random 模块从序列取出随机项

```
File: random-example-2.py



import random



# random choice from a list

for i in range(5):

    print random.choice([1, 2, 3, 5, 9])



2

3

1
```

```
9

1
```

在 2.0 及以后版本, shuffle 函数可以用于打乱一个列表的内容 (也就是生成一个该列表的随机全排列). Example 2-31 展示了如何在旧版本中实现该函数.

2.17.0.3. Example 2-31. 使用 random 模块打乱一副牌

```
File: random-example-4.py



import random



try:

    # available in 2.0 and later

    shuffle = random.shuffle

except AttributeError:

    def shuffle(x):

        for i in xrange(len(x)-1, 0, -1):

            # pick an element in x[:i+1] with which
to exchange x[i]
```

```
            j = int(random.random() * (i+1))
            x[i], x[j] = x[j], x[i]

cards = range(52)

shuffle(cards)

myhand = cards[:5]

print myhand

[4, 8, 40, 12, 30]
```

random 模块也包含了非恒定分布的随机生成器函数. Example 2-32 使用了 gauss (高斯)函数来生成满足高斯分的布随机数字.

2.17.0.4. Example 2-32. 使用 random 模块生成高斯分布随机数

```
File: random-example-3.py

```

```
import random



histogram = [0] * 20



# calculate histogram for gaussian

# noise, using average=5, stddev=1

for i in range(1000):

    i = int(random.gauss(5, 1) * 2)

    histogram[i] = histogram[i] + 1



# print the histogram

m = max(histogram)

for v in histogram:

    print "*" * (v * 50 / m)
```

****

```
**********

************************

********************************

**************************************************

**************************************************
*

************************************

**************************

**********

***

*
```

你可以在 *Python Library Reference* 找到更多关于非恒定分布随机生成器函数的信息.

标准库中提供的随机数生成器都是伪随机数生成器. 不过这对于很多目的来说已经足够了, 比如模拟, 数值分析, 以及游戏. 可以确定的是它不适合密码学用途.

2.18. whrandom 模块

这个模块早在 2.1 就被声明不赞成, 早废了. 请使用 `random` 代替.
- Feather

Example 2-33 展示了 `whrandom` , 它提供了一个伪随机数生成器. (基于 Wichmann 和 Hill, 1982 的数学运算理论). 除非你需要不共享状态的多个生成器(如多线程程序), 请使用 `random` 模块代替.

2.18.0.1. Example 2-33. 使用 whrandom 模块

```
File: whrandom-example-1.py



import whrandom



# same as random

print whrandom.random()

print whrandom.choice([1, 2, 3, 5, 9])

print whrandom.uniform(10, 20)

print whrandom.randint(100, 1000)



0.113412062346

1

16.8778954689

799
```

Example 2-34 展示了如何使用 *whrandom* 类实例创建多个生成器.

2.18.0.2. Example 2-34. 使用 whrandom 模块创建多个随机生成器

```
File: whrandom-example-2.py



import whrandom



# initialize all generators with the same seed

rand1 = whrandom.whrandom(4,7,11)

rand2 = whrandom.whrandom(4,7,11)

rand3 = whrandom.whrandom(4,7,11)



for i in range(5):

    print rand1.random(), rand2.random(),
rand3.random()
```

```
0.123993532536 0.123993532536 0.123993532536

0.180951499518 0.180951499518 0.180951499518

0.291924111809 0.291924111809 0.291924111809

0.952048889363 0.952048889363 0.952048889363
```

```
0.969794283643 0.969794283643 0.969794283643
```

2.19. md5 模块

md5 (Message-Digest Algorithm 5)模块用于计算信息密文(信息摘要).

md5 算法计算一个强壮的 128 位密文. 这意味着如果两个字符串是不同的, 那么有极高可能它们的 md5 也不同. 也就是说, 给定一个 md5 密文, 那么几乎没有可能再找到另个字符串的密文与此相同. Example 2-35 展示了如何使用 md5 模块.

2.19.0.1. Example 2-35. 使用 md5 模块

```
File: md5-example-1.py



import md5



hash = md5.new()

hash.update("spam, spam, and eggs")



print repr(hash.digest())
```

```
```

```
'L\005J\243\266\355\243u`\305r\203\267\020F\303'
```

注意这里的校验和是一个二进制字符串. Example 2-36 展示了如何获得一个十六进制或 base64 编码的字符串.

2.19.0.2. Example 2-36. 使用 md5 模块获得十六进制或 base64 编码的 md5 值

```
File: md5-example-2.py



import md5

import string

import base64



hash = md5.new()

hash.update("spam, spam, and eggs")



value = hash.digest()

print hash.hexdigest()

# before 2.0, the above can be written as
```

```
# 在 2.0 前, 以上应该写做:
```

```
# print string.join(map(lambda v: "%02x" % ord(v),
value), "")



print base64.encodestring(value)



4c054aa3b6eda37560c57283b71046c3

TAVKo7bto3VgxXKDtxBGww==
```

Example 2-37 展示了如何使用 md5 校验和来处理口令的发送与应答的验证(不过我们将稍候讨论这里使用随机数字所带来的问题).

2.19.0.3. Example 2-37. 使用 md5 模块来处理口令的发送与应答的验证

```
File: md5-example-3.py



import md5

import string, random



def getchallenge():

    # generate a 16-byte long random string.  (note
that the built-
```

```
    # in pseudo-random generator uses a 24-bit seed,
so this is not

    # as good as it may seem...)

    # 生成一个 16 字节长的随机字符串. 注意内建的
伪随机生成器

    # 使用的是 24 位的种子(seed), 所以这里这样用
并不好..

    challenge = map(lambda i: chr(random.randint(0,
255)), range(16))

    return string.join(challenge, "")



def getresponse(password, challenge):

    # calculate combined digest for password and
challenge

    # 计算密码和质询(challenge)的联合密文

    m = md5.new()

    m.update(password)

    m.update(challenge)

    return m.digest()
```

```
#

# server/client communication

# 服务器/客户端通讯



# 1. client connects.  server issues challenge.

# 1. 客户端连接, 服务器发布质询(challenge)



print "client:", "connect"



challenge = getchallenge()



print "server:", repr(challenge)



# 2. client combines password and challenge, and
calculates

# the response.

# 2. 客户端计算密码和质询(challenge)的组合后的密
文
```

```
client_response = getresponse("trustno1",
challenge)



print "client:", repr(client_response)



# 3. server does the same, and compares the result
with the

# client response.  the result is a safe login in
which the

# password is never sent across the communication
channel.

# 3. 服务器做同样的事, 然后比较结果与客户端的返
回,

# 判断是否允许用户登陆. 这样做密码没有在通讯中明
文传输.



server_response = getresponse("trustno1",
challenge)



if server_response == client_response:
```

```
    print "server:", "login ok"
```

```

```

```
client: connect

server:
'\334\352\227Z#\272\273\212KG\330\265\032>\311o'

client:
"l'\305\240-x\245\237\035\225A\254\233\337\225\00
1"

server: login ok
```

Example 2-38 提供了 md5 的一个变种, 你可以通过标记信息来判断它是否在网络传输过程中被修改(丢失).

2.19.0.4. Example 2-38. 使用 md5 模块检查数据完整性

```
File: md5-example-4.py



import md5

import array



class HMAC_MD5:

    # keyed md5 message authentication
```

```
        def _ _init_ _(self, key):

            if len(key) > 64:

                key = md5.new(key).digest()

            ipad = array.array("B", [0x36] * 64)

            opad = array.array("B", [0x5C] * 64)

            for i in range(len(key)):

                ipad[i] = ipad[i] ^ ord(key[i])

                opad[i] = opad[i] ^ ord(key[i])

            self.ipad = md5.md5(ipad.tostring())

            self.opad = md5.md5(opad.tostring())



        def digest(self, data):

            ipad = self.ipad.copy()

            opad = self.opad.copy()

            ipad.update(data)

            opad.update(ipad.digest())
```

```
        return opad.digest()
```

```

```

```
#

# simulate server end

# 模拟服务器端



key = "this should be a well-kept secret"

message = open("samples/sample.txt").read()



signature = HMAC_MD5(key).digest(message)



# (send message and signature across a public
network)

# (经过由网络发送信息和签名)



#

# simulate client end
```

```
#模拟客户端



key = "this should be a well-kept secret"


client_signature = HMAC_MD5(key).digest(message)



if client_signature == signature:
    print "this is the original message:"
    print
    print message
else:
    print "someone has modified the message!!!"
```

copy 方法会对这个内部对象状态做一个快照(snapshot). 这允许你预先计算部分密文摘要(例如 Example 2-38 中的 padded key).

该算法的细节请参阅 *HMAC-MD5:Keyed-MD5 for Message Authentication* (http://www.research.ibm.com/security/draft-ietf-ipsec-hmac-md5-00.txt) by Krawczyk, 或其他.

千万别忘记内建的伪随机生成器对于加密操作而言并不合适. 千万小心.

2.20. sha 模块

sha 模块提供了计算信息摘要(密文)的另种方法, 如 Example 2-39 所示. 它与 md5 模块类似, 但生成的是 160 位签名.

2.20.0.1. Example 2-39. 使用 sha 模块

```
File: sha-example-1.py



import sha



hash = sha.new()

hash.update("spam, spam, and eggs")



print repr(hash.digest())

print hash.hexdigest()



'\321\333\003\026I\331\272-j\303\247\240\345\343T
vq\364\346\311'

d1db031649d9ba2d6ac3a7a0e5e3547671f4e6c9
```

关于 sha 密文的使用, 请参阅 md5 中的例子.

2.21. crypt 模块

(可选, 只用于 Unix) `crypt` 模块实现了单向的 DES 加密, Unix 系统使用这个加密算法来储存密码, 这个模块真正也就只在检查这样的密码时有用.

Example 2-40 展示了如何使用 `crypt.crypt` 来加密一个密码, 将密码和 *salt* 组合起来然后传递给函数, 这里的 *salt* 包含两位随机字符. 现在你可以扔掉原密码而只保存加密后的字符串了.

2.21.0.1. Example 2-40. 使用 crypt 模块

```
File: crypt-example-1.py



import crypt



import random, string



def getsalt(chars = string.letters +
string.digits):

    # generate a random 2-character 'salt'

    # 生成随机的 2 字符 'salt'

    return random.choice(chars) +
random.choice(chars)



print crypt.crypt("bananas", getsalt())
```

```
'py8UGrijma1j6'
```

确认密码时，只需要用新密码调用加密函数，并取加密后字符串的前两位作为 *salt* 即可．如果结果和加密后字符串匹配，那么密码就是正确的．Example 2-41 使用 pwd 模块来获取已知用户的加密后密码.

2.21.0.2. Example 2-41. 使用 crypt 模块身份验证

```
File: crypt-example-2.py



import pwd, crypt



def login(user, password):

    "Check if user would be able to log in using
password"

    try:

        pw1 = pwd.getpwnam(user)[1]

        pw2 = crypt.crypt(password, pw1[:2])

        return pw1 == pw2

    except KeyError:

        return 0 # no such user
```

```
user = raw_input("username:")
password = raw_input("password:")

if login(user, password):
    print "welcome", user
else:
    print "login failed"
```

关于其他实现验证的方法请参阅 md5 模块一节.

2.22. rotor 模块

这个模块在 2.3 时被声明不赞成, 2.4 时废了. 因为它的加密算法不安全.
- Feather

(可选) rotor 模块实现了一个简单的加密算法. 如 Example 2-42 所示. 它的算法基于 WWII Enigma engine.

2.22.0.1. Example 2-42. 使用 rotor 模块

```
File: rotor-example-1.py

import rotor
```

```
SECRET_KEY = "spam"

MESSAGE = "the holy grail"



r = rotor.newrotor(SECRET_KEY)



encoded_message = r.encrypt(MESSAGE)

decoded_message = r.decrypt(encoded_message)



print "original:", repr(MESSAGE)

print "encoded message:", repr(encoded_message)

print "decoded message:", repr(decoded_message)



original: 'the holy grail'

encoded message:
'\227\271\244\015\305sw\3340\337\252\237\340U'

decoded message: 'the holy grail'
```

2.23. zlib 模块

(可选) zlib 模块为 "zlib" 压缩提供支持. (这种压缩方法是 "deflate".)

Example 2-43 展示了如何使用 compress 和 decompress 函数接受字符串参数.

2.23.0.1. Example 2-43. 使用 zlib 模块压缩字符串

```
File: zlib-example-1.py



import zlib



MESSAGE = "life of brian"



compressed_message = zlib.compress(MESSAGE)

decompressed_message =
zlib.decompress(compressed_message)



print "original:", repr(MESSAGE)
```

```
print "compressed message:",
repr(compressed_message)
```

```
print "decompressed message:",
repr(decompressed_message)
```

```
original: 'life of brian'

compressed message:
'x\234\313\311LKU\310OSH*\312L\314\003\000!\010\0
04\302'

decompressed message: 'life of brian'
```

文件的内容决定了压缩比率, Example 2-44 说明了这点.

2.23.0.2. Example 2-44. 使用 zlib 模块压缩多个不同类型文件

```
File: zlib-example-2.py

import zlib

import glob

for file in glob.glob("samples/*"):
```

```
    indata = open(file, "rb").read()
```

```
    outdata = zlib.compress(indata,
zlib.Z_BEST_COMPRESSION)




    print file, len(indata), "=>", len(outdata),

    print "%d%%" % (len(outdata) * 100 /
len(indata))
```

```
samples\sample.au 1676 => 1109 66%

samples\sample.gz 42 => 51 121%

samples\sample.htm 186 => 135 72%

samples\sample.ini 246 => 190 77%

samples\sample.jpg 4762 => 4632 97%

samples\sample.msg 450 => 275 61%

samples\sample.sgm 430 => 321 74%

samples\sample.tar 10240 => 125 1%

samples\sample.tgz 155 => 159 102%

samples\sample.txt 302 => 220 72%

samples\sample.wav 13260 => 10992 82%
```

你也可以实时地压缩或解压缩数据, 如 Example 2-45 所示.

2.23.0.3. Example 2-45. 使用 zlib 模块解压缩流

```
File: zlib-example-3.py



import zlib



encoder = zlib.compressobj()



data = encoder.compress("life")

data = data + encoder.compress(" of ")

data = data + encoder.compress("brian")

data = data + encoder.flush()



print repr(data)

print repr(zlib.decompress(data))
```

```
'x\234\313\311LKU\310OSH*\312L\314\003\000!\010\0
04\302'
```

```
'life of brian'
```

Example 2-46 把解码对象封装到了一个类似文件对象的类中，实现了一些文件对象的方法，这样使得读取压缩文件更方便.

2.23.0.4. Example 2-46. 压缩流的仿文件访问方式

```
File: zlib-example-4.py



import zlib

import string, StringIO



class ZipInputStream:



    def __init__(self, file):

        self.file = file

        self.__rewind()



    def __rewind(self):

        self.zip = zlib.decompressobj()
```

```
        self.pos = 0 # position in zipped stream

        self.offset = 0 # position in unzipped
stream

        self.data = ""



    def _ _fill(self, bytes):

        if self.zip:

            # read until we have enough bytes in the
buffer

            while not bytes or len(self.data) <
bytes:

                self.file.seek(self.pos)

                data = self.file.read(16384)

                if not data:

                    self.data = self.data +
self.zip.flush()

                    self.zip = None # no more data

                    break

                self.pos = self.pos + len(data)
```

```
                self.data = self.data +
self.zip.decompress(data)



    def seek(self, offset, whence=0):

        if whence == 0:

            position = offset

        elif whence == 1:

            position = self.offset + offset

        else:

            raise IOError, "Illegal argument"

        if position < self.offset:

            raise IOError, "Cannot seek backwards"



        # skip forward, in 16k blocks

        while position > self.offset:

            if not self.read(min(position -
self.offset, 16384)):

                break
```

```
        def tell(self):
            return self.offset

        def read(self, bytes = 0):
            self._ _fill(bytes)
            if bytes:
                data = self.data[:bytes]
                self.data = self.data[bytes:]
            else:
                data = self.data
                self.data = ""
            self.offset = self.offset + len(data)
            return data

        def readline(self):
            # make sure we have an entire line
```

```
            while self.zip and "\n" not in self.data:
```

```
                self._ _fill(len(self.data) + 512)
```

```
            i = string.find(self.data, "\n") + 1
            if i <= 0:
                return self.read()
            return self.read(i)


        def readlines(self):
            lines = []
            while 1:
                s = self.readline()
                if not s:
                    break
                lines.append(s)
            return lines


    #
```

```
# try it out
```

```
data = open("samples/sample.txt").read()

data = zlib.compress(data)



file = ZipInputStream(StringIO.StringIO(data))

for line in file.readlines():

    print line[:-1]
```

```
We will perhaps eventually be writing only small

modules which are identified by name as they are

used to build larger ones, so that devices like

indentation, rather than delimiters, might become

feasible for expressing local structure in the

source language.

    -- Donald E. Knuth, December 1974
```

2.24. code 模块

code 模块提供了一些用于模拟标准交互解释器行为的函数.

compile_command 与内建 compile 函数行为相似, 但它会通过测试来保证你传递的是一个完成的 Python 语句.

在 Example 2-47 中, 我们一行一行地编译一个程序, 编译完成后会执行所得到的代码对象 (code object). 程序代码如下:

```
a = (

  1,

  2,

  3

)

print a
```

注意只有我们到达第 2 个括号, 元组的赋值操作能编译完成.

2.24.0.1. Example 2-47. 使用 code 模块编译语句

```
File: code-example-1.py



import code

import string

```

```
#

SCRIPT = [

    "a = (",

    "  1,",

    "  2,",

    "  3 ",

    ")",

    "print a"

]



script = ""



for line in SCRIPT:

    script = script + line + "\n"

    co = code.compile_command(script, "<stdin>",
"exec")

    if co:

        # got a complete statement.  execute it!
```

```
        print "-"*40
        print script,
        print "-"*40
        exec co
        script = ""


----------------------------------------
a = (
  1,
  2,
  3
)
----------------------------------------
----------------------------------------
print a
----------------------------------------
(1, 2, 3)
```

InteractiveConsole 类实现了一个交互控制台，类似你启动的 Python 解释器交互模式.

控制台可以是活动的(自动调用函数到达下一行) 或是被动的(当有新数据时调用 *push* 方法). 默认使用内建的 `raw_input` 函数. 如果你想使用另个输入函数，你可以使用相同的名称重载这个方法. Example 2-48 展示了如何使用 `code` 模块来模拟交互解释器.

2.24.0.2. Example 2-48. 使用 code 模块模拟交互解释器

```
File: code-example-2.py



import code



console = code.InteractiveConsole()

console.interact()



Python 1.5.2

Copyright 1991-1995 Stichting Mathematisch Centrum,
Amsterdam

(InteractiveConsole)

>>> a = (

...     1,

...     2,
```

```
...     3

... )

>>> print a

(1, 2, 3)
```

Example 2-49 中的脚本定义了一个 keyboard 函数. 它允许你在程序中手动控制交互解释器.

2.24.0.3. Example 2-49. 使用 code 模块实现简单的 Debugging

```
File: code-example-3.py



def keyboard(banner=None):

    import code, sys



    # use exception trick to pick up the current
frame

    try:

        raise None

    except:

        frame = sys.exc_info()[2].tb_frame.f_back
```

```
    # evaluate commands in current namespace
    namespace = frame.f_globals.copy()
    namespace.update(frame.f_locals)

    code.interact(banner=banner, local=namespace)

def func():
    print "START"
    a = 10
    keyboard()
    print "END"

func()
```

```
START
```

Python 1.5.2

```
Copyright 1991-1995 Stichting Mathematisch Centrum,
Amsterdam

(InteractiveConsole)

>>> print a

10

>>> print keyboard

<function keyboard at 9032c8>

^Z

END
```

3. 线程和进程

"Well, since you last asked us to stop, this thread has moved from discussing languages suitable for professional programmers via accidental users to computer-phobic users. A few more iterations can make this thread really interesting..."
- eff-bot, June 1996

3.1. 概览

本章将介绍标准 Python 解释器中所提供的线程支持模块. 注意线程支持模块是可选的, 有可能在一些 Python 解释器中不可用.

本章还涵盖了一些 Unix 和 Windows 下用于执行外部进程的模块.

3.1.1. 线程

执行 Python 程序的时候，是按照从主模块顶端向下执行的. 循环用于重复执行部分代码，函数和方法会将控制临时移交到程序的另一部分.

通过线程，你的程序可以在同时处理多个任务. 每个线程都有它自己的控制流. 所以你可以在一个线程里从文件读取数据，另个向屏幕输出内容.

为了保证两个线程可以同时访问相同的内部数据，Python 使用了 *global interpreter lock (全局解释器锁)*. 在同一时间只可能有一个线程执行 Python 代码；Python 实际上是自动地在一段很短的时间后切换到下个线程执行，或者等待 一个线程执行一项需要时间的操作(例如等待通过 socket 传输的数据，或是从文件中读取数据).

全局锁事实上并不能避免你程序中的问题. 多个线程尝试访问相同的数据会导致异常 状态. 例如以下的代码:

```
def getitem(key):

    item = cache.get(key)

    if item is None:

        # not in cache; create a new one

        item = create_new_item(key)

     cache[key] = item

    return item
```

如果不同的线程先后使用相同的 key 调用这里的 `getitem` 方法，那么它们很可能会导致相同的参数调用两次 `create_new_item` . 大多时候这样做没有问题，但在某些时候会导致严重错误.

不过你可以使用 *lock objects* 来同步线程. 一个线程只能拥有一个 *lock object* ，这样就可以确保某个时刻 只有一个线程执行 *getitem* 函数.

3.1.2. 进程

在大多现代操作系统中，每个程序在它自身的*进程(process)*内执行. 我们通过在 shell 中键入命令或直接在菜单中选择来执行一个程序/进程. Python 允许你在一个脚本内执行一个新的程序.

大多进程相关函数通过 os 模块定义. 相关内容请参阅 第 1.4.4 小节 .

3.2. threading 模块

(可选) threading 模块为线程提供了一个高级接口，如 Example 3-1 所示. 它源自 Java 的线程实现. 和低级的 thread 模块相同，只有你在编译解释器时打开了线程支持才可以使用它 .

你只需要继承 *Thread* 类，定义好 run 方法，就可以创建一 个新的线程. 使用时首先创建该类的一个或多个实例，然后调用 start 方法. 这样每个实例的 run 方法都会运行在它自己的线程里.

3.2.0.1. Example 3-1. 使用 threading 模块

```
File: threading-example-1.py



import threading

import time, random



class Counter:
```

```
    def _ _init_ _(self):
```

```
        self.lock = threading.Lock()

        self.value = 0
```

```
    def increment(self):
        self.lock.acquire() # critical section
        self.value = value = self.value + 1
        self.lock.release()
        return value

counter = Counter()

class Worker(threading.Thread):

    def run(self):
        for i in range(10):
            # pretend we're doing something that
takes 10?00 ms
            value = counter.increment() # increment
global counter
```

```
            time.sleep(random.randint(10, 100) /
1000.0)

            print self.getName(), "-- task", i,
"finished", value



#

# try it



for i in range(10):

    Worker().start() # start a worker



Thread-1 -- task 0 finished 1

Thread-3 -- task 0 finished 3

Thread-7 -- task 0 finished 8

Thread-1 -- task 1 finished 7

Thread-4 -- task 0 Thread-5 -- task 0 finished 4

finished 5

Thread-8 -- task 0 Thread-6 -- task 0 finished 9
```

```
finished 6

...

Thread-6 -- task 9 finished 98

Thread-4 -- task 9 finished 99

Thread-9 -- task 9 finished 100
```

Example 3-1 使用了 *Lock* 对象来在全局 *Counter* 对象里创建临界区 (critical section). 如果删除了 `acquire` 和 `release` 语句, 那么 `Counter` 很可能不会到达 100.

3.3. Queue 模块

`Queue` 模块提供了一个线程安全的队列 (queue) 实现, 如 Example 3-2 所示. 你可以通过它在多个线程里安全访问同个对象.

3.3.0.1. Example 3-2. 使用 Queue 模块

```
File: queue-example-1.py



import threading

import Queue
```

```
import time, random
```

```

```

```
WORKERS = 2



class Worker(threading.Thread):



    def _ _init_ _(self, queue):

        self._ _queue = queue

        threading.Thread._ _init_ _(self)



    def run(self):

        while 1:

            item = self._ _queue.get()

            if item is None:

                break # reached end of queue



            # pretend we're doing something that
takes 10?00 ms

            time.sleep(random.randint(10, 100) /
1000.0)
```

```
            print "task", item, "finished"


#
# try it

queue = Queue.Queue(0)


for i in range(WORKERS):
    Worker(queue).start() # start a worker


for i in range(10):
    queue.put(i)


for i in range(WORKERS):
    queue.put(None) # add end-of-queue markers
```

```
task 1 finished

task 0 finished

task 3 finished

task 2 finished

task 4 finished

task 5 finished

task 7 finished

task 6 finished

task 9 finished

task 8 finished
```

Example 3-3 展示了如何限制队列的大小. 如果队列满了, 那么控制主线程 (producer threads) 被阻塞, 等待项目被弹出 (pop off).

3.3.0.2. Example 3-3. 使用限制大小的 Queue 模块

```
File: queue-example-2.py



import threading

import Queue
```

```
import time, random


WORKERS = 2


class Worker(threading.Thread):


    def _ _init_ _(self, queue):

        self._ _queue = queue

        threading.Thread._ _init_ _(self)


    def run(self):

        while 1:

            item = self._ _queue.get()

            if item is None:

                break # reached end of queue
```

```
            # pretend we're doing something that
takes 10?00 ms

            time.sleep(random.randint(10, 100) /
1000.0)



            print "task", item, "finished"



#

# run with limited queue



queue = Queue.Queue(3)



for i in range(WORKERS):

    Worker(queue).start() # start a worker



for item in range(10):

    print "push", item

    queue.put(item)
```

```
for i in range(WORKERS):
    queue.put(None) # add end-of-queue markers
```

```
push 0
push 1
push 2
push 3
push 4
push 5
task 0 finished
push 6
task 1 finished
push 7
task 2 finished
push 8
task 3 finished
```

```
push 9

task 4 finished

task 6 finished

task 5 finished

task 7 finished

task 9 finished

task 8 finished
```

你可以通过继承 *Queue* 类来修改它的行为. Example 3-4 为我们展示了一个简单的具有优先级的队列. 它接受一个元组作为参数, 元组的第一个成员表示优先级(数值越小优先级越高).

3.3.0.3. Example 3-4. 使用 Queue 模块实现优先级队列

```
File: queue-example-3.py



import Queue

import bisect



Empty = Queue.Empty


```

```
class PriorityQueue(Queue.Queue):

    "Thread-safe priority queue"



    def _put(self, item):

        # insert in order

        bisect.insort(self.queue, item)



#

# try it



queue = PriorityQueue(0)



# add items out of order

queue.put((20, "second"))

queue.put((10, "first"))

queue.put((30, "third"))
```

```
# print queue contents

try:

    while 1:

        print queue.get_nowait()

except Empty:

    pass



third

second

first
```

Example 3-5 展示了一个简单的堆栈（stack）实现（末尾添加，头部弹出，而非头部添加，头部弹出）.

3.3.0.4. Example 3-5. 使用 Queue 模块实现一个堆栈

```
File: queue-example-4.py



import Queue
```

```
    Empty = Queue.Empty
```

```

```

```
    class Stack(Queue.Queue):

        "Thread-safe stack"



        def _put(self, item):

            # insert at the beginning of queue, not at
    the end

            self.queue.insert(0, item)



        # method aliases

        push = Queue.Queue.put

        pop = Queue.Queue.get

        pop_nowait = Queue.Queue.get_nowait



    #

    # try it
```

```
stack = Stack(0)
```

```

```

```
# push items on stack

stack.push("first")

stack.push("second")

stack.push("third")



# print stack contents

try:

    while 1:

        print stack.pop_nowait()

except Empty:

    pass
```

```
third

second
```

```
first
```

3.4. thread 模块

(可选) thread 模块提为线程提供了一个低级 (low_level) 的接口, 如 Example 3-6 所示. 只有你在编译解释器时打开了线程支持才可以使用它. 如果没有特殊需要, 最好使用高级接口 threading 模块替代.

3.4.0.1. Example 3-6. 使用 thread 模块

```
File: thread-example-1.py



import thread

import time, random



def worker():

    for i in range(50):

        # pretend we're doing something that takes
10±00 ms

        time.sleep(random.randint(10, 100) /
1000.0)

        print thread.get_ident(), "-- task", i,
"finished"
```

```
#
```

```
# try it out!


for i in range(2):
    thread.start_new_thread(worker, ())


time.sleep(1)


print "goodbye!"
```

```
311 -- task 0 finished
265 -- task 0 finished
265 -- task 1 finished
311 -- task 1 finished
...
265 -- task 17 finished
```

```
311 -- task 13 finished
```

```
265 -- task 18 finished
```

```
goodbye!
```

注意当主程序退出的时候, 所有的线程也随着退出. 而 threading 模块不存在这个问题 . (该行为可改变)

3.5. commands 模块

(只用于 Unix) commands 模块包含一些用于执行外部命令的函数. Example 3-7 展示了这个模块.

3.5.0.1. Example 3-7. 使用 commands 模块

```
File: commands-example-1.py



import commands



stat, output = commands.getstatusoutput("ls -lR")



print "status", "=>", stat

print "output", "=>", len(output), "bytes"
```

```
```

```
status => 0
```

```
output => 171046 bytes
```

3.6. pipes 模块

(只用于 Unix) pipes 模块提供了 "转换管道 (conversion pipelines)" 的支持. 你可以创建包含许多外部工具调用的管道来处理多个文件. 如 Example 3-8 所示.

3.6.0.1. Example 3-8. 使用 pipes 模块

```
File: pipes-example-1.py



import pipes



t = pipes.Template()



# create a pipeline

# 这里 " - " 代表从标准输入读入内容

t.append("sort", "--")
```

```
t.append("uniq", "--")
```

```

```

```
# filter some text

# 这里空字符串代表标准输出

t.copy("samples/sample.txt", "")
```

```
Alan Jones (sensible party)

Kevin Phillips-Bong (slightly silly)

Tarquin
Fin-tim-lin-bin-whin-bim-lin-bus-stop-F'tang-F'ta
ng-Olé-Biscuitbarrel
```

3.7. popen2 模块

popen2 模块允许你执行外部命令，并通过流来分别访问它的 stdin 和 stdout （ 可能还有 stderr ）.

在 python 1.5.2 以及之前版本，该模块只存在于 Unix 平台上. 2.0 后，Windows 下也实现了该函数. Example 3-9 展示了如何使用该模块来给字符串排序.

3.7.0.1. Example 3-9. 使用 popen2 模块对字符串排序 Module to Sort Strings

```
File: popen2-example-1.py
```

```
import popen2, string

fin, fout = popen2.popen2("sort")

fout.write("foo\n")
fout.write("bar\n")
fout.close()

print fin.readline(),
print fin.readline(),
fin.close()

bar
foo
```

Example 3-10 展示了如何使用该模块控制应用程序 .

3.7.0.2. Example 3-10. 使用 popen2 模块控制 gnuchess

File: popen2-example-2.py

```
import popen2

import string



class Chess:

    "Interface class for chesstool-compatible
programs"



    def _ _init_ _(self, engine = "gnuchessc"):

        self.fin, self.fout =
popen2.popen2(engine)

        s = self.fin.readline()

        if s != "Chess\n":

            raise IOError, "incompatible chess
program"



    def move(self, move):

        self.fout.write(move + "\n")
```

```
            self.fout.flush()

            my = self.fin.readline()

            if my == "Illegal move":

                raise ValueError, "illegal move"

            his = self.fin.readline()

            return string.split(his)[2]



        def quit(self):

            self.fout.write("quit\n")

            self.fout.flush()



    #

    # play a few moves



    g = Chess()



    print g.move("a2a4")
```

```
print g.move("b2b3")



g.quit()



b8c6

e7e5
```

3.8. signal 模块

你可以使用 signal 模块配置你自己的信号处理器 (signal handler), 如 Example 3-11 所示. 当解释器收到某个信号时, 信号处理器会立即执行.

3.8.0.1. Example 3-11. 使用 signal 模块

```
File: signal-example-1.py



import signal

import time



def handler(signo, frame):
```

```
    print "got signal", signo



signal.signal(signal.SIGALRM, handler)



# wake me up in two seconds

signal.alarm(2)



now = time.time()



time.sleep(200)



print "slept for", time.time() - now, "seconds"



got signal 14

slept for 1.99262607098 seconds
```

4. 数据表示

"PALO ALTO, Calif. - Intel says its Pentium Pro and new Pentium II chips have a flaw that can cause computers to sometimes make mistakes but said the problems could be fixed easily with rewritten software."
- Reuters telegram

4.1. 概览

本章描述了一些用于在 Python 对象和其他数据表示类型间相互转换的模块. 这些模块通常用于读写特定的文件格式或是储存/取出 Python 变量.

4.1.1. 二进制数据

Python 提供了一些用于二进制数据解码/编码的模块. `struct` 模块用于在二进制数据结构(例如 C 中的 struct)和 Python 元组间转换. `array` 模块将二进制数据阵列 (C arrays)封装为 Python 序列对象.

4.1.2. 自描述格式

`marshal` 和 `pickle` 模块用于在不同的 Python 程序间共享/传递数据.

`marshal` 模块使用了简单的自描述格式(Self-Describing Formats), 它支持大多的内建数据类型, 包括 code 对象. Python 自身也使用了这个格式来储存编译后代码(.pyc 文件).

`pickle` 模块提供了更复杂的格式, 它支持用户定义的类, 自引用数据结构等等. `pickle` 是用 Python 写的, 相对来说速度较慢, 不过还有一个 `cPickle` 模块, 使用 C 实现了相同的功能, 速度和 `marshal` 不相上下.

4.1.3. 输出格式

一些模块提供了增强的格式化输出, 用来补充内建的 `repr` 函数和 `%` 字符串格式化操作符.

`pprint` 模块几乎可以将任何 Python 数据结构很好地打印出来(提高可读性).

`repr` 模块可以用来替换内建同名函数. 该模块与内建函数不同的是它限制了很多输出形式: 他只会输出字符串的前 30 个字符, 它只打印嵌套数据结构的几个等级, 等等.

4.1.4. 编码二进制数据

Python 支持大部分常见二进制编码，例如 base64 ， binhex （一种 Macintosh 格式），quoted printable ， 以及 uu 编码.

4.2. array 模块

array 模块实现了一个有效的阵列储存类型. 阵列和列表类似，但其中所有的项目必须为相同的类型. 该类型在阵列创建时指定.

Examples 4-1 到 4-5 都是很简单的范例. Example 4-1 创建了一个 *array* 对象，然后使用 tostring 方法将内部缓冲区(internal buffer)复制到字符串.

4.2.0.1. Example 4-1. 使用 array 模块将数列转换为字符串

```
File: array-example-1.py



import array



a = array.array("B", range(16)) # unsigned char
```

```
b = array.array("h", range(16)) # signed short
```

```

```

```
print a

print repr(a.tostring())
```

```
print b

print repr(b.tostring())




array('B', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15])

'\000\001\002\003\004\005\006\007\010\011\012\013
\014\015\016\017'




array('h', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15])

'\000\000\001\000\002\000\003\000\004\000\005\000
\006\000\007\000

\010\000\011\000\012\000\013\000\014\000\015\000\
016\000\017\000'
```

array 对象可以作为一个普通列表对待, 如 Example 4-2 所示. 不过, 你不能连接两个不同类型的阵列.

4.2.0.2. Example 4-2. 作为普通序列操作阵列

```
File: array-example-2.py



import array
```

```
a = array.array("B", [1, 2, 3])



a.append(4)



a = a + a



a = a[2:-2]



print a

print repr(a.tostring())

for i in a:

    print i,
```

```
array('B', [3, 4, 1, 2])
```

```
'\003\004\001\002'

3 4 1 2
```

该模块还提供了用于转换原始二进制数据到整数序列(或浮点数数列, 具体情况决定)的方法, 如 Example 4-3 所示.

4.2.0.3. Example 4-3. 使用阵列将字符串转换为整数列表

```
File: array-example-3.py



import array



a = array.array("i", "fish license") # signed
integer



print a

print repr(a.tostring())

print a.tolist()
```

```
array('i', [1752394086, 1667853344, 1702063717])
```

```
'fish license'
```

```
[1752394086, 1667853344, 1702063717]
```

最后，Example 4-4 展示了如何使用该模块判断当前平台的字节序（ endianess ）.

4.2.0.4. Example 4-4. 使用 array 模块判断平台字节序

```
File: array-example-4.py



import array



def little_endian():

    return
ord(array.array("i",[1]).tostring()[0])



if little_endian():

    print "little-endian platform (intel, alpha)"

else:

    print "big-endian platform (motorola, sparc)"
```

```

```

```
big-endian platform (motorola, sparc)
```

Python 2.0 以及以后版本提供了 sys.byteorder 属性，可以更简单地判断字节序（属性值为 "little" 或 "big"），如 Example 4-5 所示.

4.2.0.5. Example 4-5. 使用 sys.byteorder 属性判断平台字节序(Python 2.0 及以后)

```
File: sys-byteorder-example-1.py



import sys



# 2.0 and later

if sys.byteorder == "little":

    print "little-endian platform (intel, alpha)"

else:

    print "big-endian platform (motorola, sparc)"



big-endian platform (motorola, sparc)
```

4.3. struct 模块

struct 模块用于转换二进制字符串和 Python 元组. pack 函数接受格式字符串以及额外参数，根据指定格式将额外参数转换为二进制字符串. upack 函数接受一个字符串作为参数，返回一个元组. 如 Example 4-6 所示.

4.3.0.1. Example 4-6. 使用 struct 模块

```
File: struct-example-1.py



import struct



# native byteorder

buffer = struct.pack("ihb", 1, 2, 3)

print repr(buffer)

print struct.unpack("ihb", buffer)



# data from a sequence, network byteorder

data = [1, 2, 3]

buffer = apply(struct.pack, ("!ihb",) +
tuple(data))
```

```
print repr(buffer)
```

```
print struct.unpack("!ihb", buffer)
```

```
# in 2.0, the apply statement can also be written as:

# buffer = struct.pack("!ihb", *data)



'\001\000\000\000\002\000\003'

(1, 2, 3)

'\000\000\000\001\000\002\003'

(1, 2, 3)
```

4.4. xdrlib 模块

xdrlib 模块用于在 Python 数据类型和 Sun 的 external data representation (XDR) 间相互转化, 如 Example 4-7 所示.

4.4.0.1. Example 4-7. 使用 xdrlib 模块

```
File: xdrlib-example-1.py



import xdrlib
```

```


#
```

```
# create a packer and add some data to it



p = xdrlib.Packer()

p.pack_uint(1)

p.pack_string("spam")



data = p.get_buffer()



print "packed:", repr(data)



#

# create an unpacker and use it to decode the data



u = xdrlib.Unpacker(data)
```

```

```

```
print "unpacked:", u.unpack_uint(),
repr(u.unpack_string())
```

```
u.done()



packed: '\000\000\000\001\000\000\000\004spam'

unpacked: 1 'spam'
```

Sun 在 remote procedure call (RPC) 协议中使用了 XDR 格式. Example 4-8 虽然不完整, 但它展示了如何建立一个 RPC 请求包.

4.4.0.2. Example 4-8. 使用 xdrlib 模块发送 RPC 调用包

```
File: xdrlib-example-2.py



import xdrlib



# some constants (see the RPC specs for details)

RPC_CALL = 1

RPC_VERSION = 2
```

```

```

```
MY_PROGRAM_ID = 1234 # assigned by Sun
```

```
MY_VERSION_ID = 1000

MY_TIME_PROCEDURE_ID = 9999



AUTH_NULL = 0



transaction = 1



p = xdrlib.Packer()



# send a Sun RPC call package

p.pack_uint(transaction)

p.pack_enum(RPC_CALL)

p.pack_uint(RPC_VERSION)

p.pack_uint(MY_PROGRAM_ID)

p.pack_uint(MY_VERSION_ID)

p.pack_uint(MY_TIME_PROCEDURE_ID)
```

```
p.pack_enum(AUTH_NULL)

p.pack_uint(0)

p.pack_enum(AUTH_NULL)

p.pack_uint(0)



print repr(p.get_buffer())



'\000\000\000\001\000\000\000\001\000\000\000\002
\000\000\004\322

\000\000\003\350\000\000\'\017\000\000\000\000\00
0\000\000\000\000

\000\000\000\000\000\000\000'
```

4.5. marshal 模块

marshal 模块可以把不连续的数据组合起来 - 与字符串相互转化，这样它们就可以写入文件或是在网络中传输. 如 Example 4-9 所示.

marshal 模块使用了简单的自描述格式. 对于每个数据项目，格式化后的字符串都包含一个类型代码，然后是一个或多个类型标识区域. 整数使用小字节序(little-endian order)储存，字符串储存时和它自身内容长度相同(可能包含空字节)，元组由组成它的对象组合表示.

4.5.0.1. Example 4-9. 使用 marshal 模块组合不连续数据

```
File: marshal-example-1.py



import marshal



value = (

    "this is a string",

    [1, 2, 3, 4],

    ("more tuples", 1.0, 2.3, 4.5),

    "this is yet another string"

    )



data = marshal.dumps(value)



# intermediate format

print type(data), len(data)



print "-"*50
```

```
print repr(data)

print "-"*50



print marshal.loads(data)



<type 'string'> 118

----------------------------------------------
-

'(\004\000\000\000s\020\000\000\000this is a
string

[\004\000\000\000i\001\000\000\000i\002\000\000\0
00

i\003\000\000\000i\004\000\000\000(\004\000\000\0
00

s\013\000\000\000more
tuplesf\0031.0f\0032.3f\0034.

5s\032\000\000\000this is yet another string'

----------------------------------------------
-

('this is a string', [1, 2, 3, 4], ('more tuples',
```

```
    1.0, 2.3, 4.5), 'this is yet another string')
```

marshal 模块还可以处理 code 对象(它用于储存预编译的 Python 模块). 如 Example 4-10 所示.

4.5.0.2. Example 4-10. 使用 marshal 模块处理代码

```
File: marshal-example-2.py



import marshal



script = """

print 'hello'

"""



code = compile(script, "<script>", "exec")



data = marshal.dumps(code)



# intermediate format
```

```
print type(data), len(data)



print "-"*50

print repr(data)

print "-"*50



exec marshal.loads(data)



<type 'string'> 81

----------------------------------------------
-

'c\000\000\000\000\001\000\000\000s\017\000\000\0
0

0\177\000\000\177\002\000d\000\000GHd\001\000S(\0
0

2\000\000\000s\005\000\000\000helloN(\000\000\000
\

000(\000\000\000\000s\010\000\000\000<script>s\00
1

\000\000\000?\002\000s\000\000\000\000'
```

```
------------------------------------------------
-

hello
```

4.6. pickle 模块

pickle 模块同 marshal 模块相同，将数据连续化，便于保存传输. 它比 marshal 要慢一些，但它可以处理类实例，共享的元素，以及递归数据结构等.

4.6.0.1. Example 4-11. 使用 pickle 模块

```
File: pickle-example-1.py



import pickle



value = (

    "this is a string",

    [1, 2, 3, 4],

    ("more tuples", 1.0, 2.3, 4.5),

    "this is yet another string"
```

```
    )



data = pickle.dumps(value)



# intermediate format

print type(data), len(data)



print "-"*50

print data

print "-"*50



print pickle.loads(data)



<type 'string'> 121

----------
-

(S'this is a string'

p0
```

```
(lp1

I1

aI2

aI3

aI4

a(S'more tuples'

p2

F1.0

F2.3

F4.5

tp3

S'this is yet another string'

p4

tp5

.

----------------------------------------------
-

('this is a string', [1, 2, 3, 4], ('more tuples',
```

```
    1.0, 2.3, 4.5), 'this is yet another string')
```

不过另一方面, pickle 不能处理 code 对象(可以参阅 copy_reg 模块来完成这个).

默认情况下, pickle 使用急于文本的格式. 你也可以使用二进制格式, 这样数字和二进制字符串就会以紧密的格式储存, 这样文件就会更小点. 如 Example 4-12 所示.

4.6.0.2. Example 4-12. 使用 pickle 模块的二进制模式

```
File: pickle-example-2.py



import pickle

import math



value = (

    "this is a long string" * 100,

    [1.2345678, 2.3456789, 3.4567890] * 100

    )



# text mode

data = pickle.dumps(value)
```

```
print type(data), len(data), pickle.loads(data) ==
value



# binary mode

data = pickle.dumps(value, 1)

print type(data), len(data), pickle.loads(data) ==
value
```

4.7. cPickle 模块

(可选, 注意大小写) cPickle 模块是针对 pickle 模块的一个更快的实现. 如 Example 4-13 所示.

4.7.0.1. Example 4-13. 使用 cPickle 模块

```
File: cpickle-example-1.py



try:

    import cPickle

    pickle = cPickle

except ImportError:

    import pickle
```

4.8. copy_reg 模块

你可以使用 copy_reg 模块注册你自己的扩展类型. 这样 pickle 和 copy 模块就会知道如何处理非标准类型.

例如, 标准的 pickle 实现不能用来处理 Python code 对象, 如下所示:

```
File: copy-reg-example-1.py



import pickle



CODE = """

print 'good evening'

"""



code = compile(CODE, "<string>", "exec")



exec code

exec pickle.loads(pickle.dumps(code))
```

```
good evening

Traceback (innermost last):

...

pickle.PicklingError: can't pickle 'code' objects
```

我们可以注册一个 code 对象处理器来完成目标. 处理器应包含两个部分: 一个 pickler , 接受 code 对象并返回一个只包含简单数据类型的元组, 以及一个 unpickler , 作用相反, 接受这样的元组作为参数. 如 Example 4-14 所示.

4.8.0.1. Example 4-14. 使用 copy_reg 模块实现 code 对象的 pickle 操作

```
File: copy-reg-example-2.py



import copy_reg

import pickle, marshal, types



#

# register a pickle handler for code objects



def code_unpickler(data):
```

```
    return marshal.loads(data)



def code_pickler(code):

    return code_unpickler, (marshal.dumps(code),)



copy_reg.pickle(types.CodeType, code_pickler,
code_unpickler)



#

# try it out



CODE = """

print "suppose he's got a pointed stick"

"""



code = compile(CODE, "<string>", "exec")
```

```
exec code

exec pickle.loads(pickle.dumps(code))



suppose he's got a pointed stick

suppose he's got a pointed stick
```

如果你是在网络中传输 pickle 后的数据, 那么请确保自定义的 unpickler 在数据接收端也是可用的.

Example 4-15 展示了如何实现 pickle 一个打开的文件对象.

4.8.0.2. Example 4-15. 使用 copy_reg 模块实现文件对象的 pickle 操作

```
File: copy-reg-example-3.py



import copy_reg

import pickle, types

import StringIO



#

# register a pickle handler for file objects
```

```
def file_unpickler(position, data):

    file = StringIO.StringIO(data)

    file.seek(position)

    return file



def file_pickler(code):

    position = file.tell()

    file.seek(0)

    data = file.read()

    file.seek(position)

    return file_unpickler, (position, data)



copy_reg.pickle(types.FileType, file_pickler,
file_unpickler)



#

# try it out
```

```
file = open("samples/sample.txt", "rb")



print file.read(120),

print "<here>",

print pickle.loads(pickle.dumps(file)).read()



We will perhaps eventually be writing only small

modules, which are identified by name as they are

used to build larger <here> ones, so that devices
like

indentation, rather than delimiters, might become

feasible for expressing local structure in the

source language.

     -- Donald E. Knuth, December 1974
```

4.9. pprint 模块

pprint 模块(pretty printer)用于打印 Python 数据结构. 当你在命令行下打印特定数据结构时你会发现它很有用(输出格式比较整齐, 便于阅读).

4.9.0.1. Example 4-16. 使用 pprint 模块

```
File: pprint-example-1.py



import pprint



data = (

    "this is a string", [1, 2, 3, 4], ("more tuples",

    1.0, 2.3, 4.5), "this is yet another string"

    )



pprint.pprint(data)
```

```
('this is a string',

 [1, 2, 3, 4],

 ('more tuples', 1.0, 2.3, 4.5),

 'this is yet another string')
```

4.10. repr 模块

repr 模块提供了内建 repr 函数的另个版本. 它限制了很多(字符串长度, 递归等). Example 4-17 展示了如何使用该模块.

4.10.0.1. Example 4-17. 使用 repr 模块

```
File: repr-example-1.py



# note: this overrides the built-in 'repr' function

from repr import repr



# an annoyingly recursive data structure

data = (

    "X" * 100000,

    )

data = [data]

data.append(data)



print repr(data)
```

```
[('XXXXXXXXXXXX...XXXXXXXXXXXXX',),
[('XXXXXXXXXXXX...XXXXXXXXXX

XXX',), [('XXXXXXXXXXXX...XXXXXXXXXXXXX',),
[('XXXXXXXXXXXX...XX

XXXXXXXXXX',),
[('XXXXXXXXXXXX...XXXXXXXXXXXXX',), [(...), [...

]]]]]]]
```

4.11. base64 模块

base64 编码体系用于将任意二进制数据转换为纯文本. 它将一个 3 字节的二进制字节组转换为 4 个文本字符组储存, 而且规定只允许以下集合中的字符出现:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

0123456789+/
```

另外, = 用于填充数据流的末尾.

Example 4-18 展示了如何使用 encode 和 decode 函数操作文件对象.

4.11.0.1. Example 4-18. 使用 base64 模块编码文件

```
File: base64-example-1.py
```

```
import base64



MESSAGE = "life of brian"



file = open("out.txt", "w")

file.write(MESSAGE)

file.close()



base64.encode(open("out.txt"), open("out.b64",
"w"))

base64.decode(open("out.b64"), open("out.txt",
"w"))



print "original:", repr(MESSAGE)

print "encoded message:",
repr(open("out.b64").read())

print "decoded message:",
repr(open("out.txt").read())
```

```
original:  'life of brian'

encoded message: 'bGlmZSBvZiBicmlhbg==\012'

decoded message: 'life of brian'
```

Example 4-19 展示了如何使用 encodestring 和 decodestring 函数在字符串间转换. 它们是 encode 和 decode 函数的顶层封装. 使用 StringIO 对象处理输入和输出.

4.11.0.2. Example 4-19. 使用 base64 模块编码字符串

```
File: base64-example-2.py



import base64



MESSAGE = "life of brian"



data = base64.encodestring(MESSAGE)



original_data = base64.decodestring(data)



print "original:", repr(MESSAGE)
```

```
print "encoded data:", repr(data)

print "decoded data:", repr(original_data)



original: 'life of brian'

encoded data: 'bGlmZSBvZiBicmlhbg==\012'

decoded data: 'life of brian'
```

Example 4-20 展示了如何将用户名和密码转换为 HTTP 基本身份验证字符串.

4.11.0.3. Example 4-20. 使用 base64 模块做基本验证

```
File: base64-example-3.py



import base64



def getbasic(user, password):

    # basic authentication (according to HTTP)

    return base64.encodestring(user + ":" +
password)
```

```
print getbasic("Aladdin", "open sesame")



'QWxhZGRpbjpvcGVuIHNlc2FtZQ=='
```

最后, Example 4-21 展示了一个实用小工具, 它可以把 GIF 格式转换为 Python 脚本, 便于使用 Tkinter 库.

4.11.0.4. Example 4-21. 使用 base64 为 Tkinter 封装 GIF 格式

```
File: base64-example-4.py



import base64, sys



if not sys.argv[1:]:

    print "Usage: gif2tk.py giffile >pyfile"

    sys.exit(1)



data = open(sys.argv[1], "rb").read()



if data[:4] != "GIF8":
```

```
    print sys.argv[1], "is not a GIF file"
```

```
    sys.exit(1)
```

```
print '# generated from', sys.argv[1], 'by
gif2tk.py'

print

print 'from Tkinter import PhotoImage'

print

print 'image = PhotoImage(data="""'

print base64.encodestring(data),

print '""")'



# generated from samples/sample.gif by gif2tk.py



from Tkinter import PhotoImage



image = PhotoImage(data="""
```

```
R0lGODlhoAB4APcAAAAAAIAAAACAAICAAAAAgIAAgACAgICAg
AQEBIwEBIyMBJRUlISE/LRUBAQE

...

AjmQBFmQBnmQCJmQCrmQDNmQDvmQEBmREnkRAQEAOw==

""")
```

4.12. binhex 模块

binhex 模块用于到 Macintosh BinHex 格式的相互转化. 如 Example 4-22 所示.

4.12.0.1. Example 4-22. 使用 binhex 模块

```
File: binhex-example-1.py



import binhex

import sys



infile = "samples/sample.jpg"



binhex.binhex(infile, sys.stdout)
```

```
(This file must be converted with BinHex 4.0)
```

```
:#ROKEA"XC5jUF'F!2j!)!*!%%TS!N!4RdrrBrq!!%%T'58B!
!3%!!!%!!3!!rpX

!3`!)"JB("J8)"`F(#3N)#J`8$3`,#``C%K-2&"dD(aiG'K`F
)#3Z*b!L,#-F(#J

h+5``-63dO"mR16di-M`Z-c3brpX!3`%*#3N-#``B$3dB-L%F
)6+3-[r!!"%)!)!

!J!-")J!#%3%$%3(ra!!I!!!""3'3"J#3#!%#!`3&"JF)#3S,
rm3!Y4!!!J%$!`)

%!`8&"!3!!!&p!3)$!!34"4)K-8%'%e&K"b*a&$+"ND%))d+a
`495dI!N-f*bJJN
```

该模块有两个函数 binhex 和 hexbin .

4.13. quopri 模块

quopri 模块基于 MIME 标准实现了引用的可打印编码(quoted printable encoding).

这样的编码可以将不包含或只包含一部分 U.S. ASCII 文本的信息, 例如大多欧洲语言, 中文, 转换为只包含 U.S. ASCII 的信息. 在一些老式的 mail 代理中你会发现这很有用, 因为它们一般不支持特殊. 如 Example 4-23 所示.

4.13.0.1. Example 4-23. 使用 quopri 模块

File: quopri-example-1.py

```
import quopri
```

```
import StringIO



# helpers (the quopri module only supports
file-to-file conversion)



def encodestring(instring, tabs=0):

    outfile = StringIO.StringIO()

    quopri.encode(StringIO.StringIO(instring),
outfile, tabs)

    return outfile.getvalue()



def decodestring(instring):

    outfile = StringIO.StringIO()

    quopri.decode(StringIO.StringIO(instring),
outfile)

    return outfile.getvalue()
```

```
#
```

```
# try it out
```

```
MESSAGE = "å i åa ä e ö!"


encoded_message = encodestring(MESSAGE)

decoded_message = decodestring(encoded_message)


print "original:", MESSAGE

print "encoded message:", repr(encoded_message)

print "decoded message:", decoded_message


original: å i åa ä e ö!

encoded message: '=E5 i =E5a =E4 e =F6!\012'

decoded message: å i åa ä e ö!
```

如 Example 4-23 所示，非 U.S. 字符通过等号 (=) 附加两个十六进制字符来表示. 这里需要注意等号也是使用这样的方式("=3D")来表示的，以及换行符("=20"). 其他字符不会被改变. 所以如果你没有用太多的怪异字符的话，编码后字符串依然可读性很好.

(Europeans generally hate this encoding and strongly believe that certain U.S. programmers deserve to be slapped in the head with a huge great fish to the jolly music of Edward German....)

4.14. uu 模块

uu 编码体系用于将任意二进制数据转换为普通文本格式. 该格式在新闻组中很流行，但逐渐被 base64 编码取代.

uu 编码将每个 3 字节(24 位)的数据组转换为 4 个可打印字符(每个字符 6 位)，使用从 chr(32) (空格) 到 chr(95) 的字符. uu 编码通常会使数据大小增加 40% .

一个编码后的数据流以一个新行开始，它包含文件的权限(Unix 格式)和文件名，以 end 行结尾:

```
begin 666 sample.jpg

M_]C_X  02D9)1@ ! 0   0 !  #_VP!#
@&!@<&!0@'!P<)'0@*#!0-# L+

...more lines like this...

end
```

uu 模块提供了两个函数: encode 和 decode .

encode(infile, outfile, filename) 函数从编码输入文件中的数据，然后写入到输出文件中. 如 Example 4-24 所示. infile 和 outfile 可以是文件名或文件对象. filename 参数作为起始域的文件名写入.

4.14.0.1. Example 4-24. 使用 uu 模块编码二进制文件

```
File: uu-example-1.py
```

```
import uu

import os, sys



infile = "samples/sample.jpg"



uu.encode(infile, sys.stdout,
os.path.basename(infile))



begin 666 sample.jpg

M_]C_X  02D9)1@ ! 0   0 !  #_VP!#
@&!@<&!0@'!P<)"0@*#!0-# L+

M#!D2$P\4'1H?'AT:'!P@)"XG("(L(QP<*#<I+#
Q-#0T'R<Y/3@R/"XS-#+_

MVP!#
0D)"0P+#!@-#1@R(1PA,C(R,C(R,C(R,C(R,C(R,C(R,C(R,C
(R,C(R

M,C(R,C(R,C(R,C(R,C(R,C(R,C(R,C+_P  1" "  ( #
2(  A$! Q$!_\0

M'P   04! 0$! 0$           $" P0%!@<("0H+_\0 M1
@$# P($ P4%
```

decode(infile, outfile) 函数用来解码 uu 编码的数据. 同样地, 参数可以是文件名也可以是文件对象. 如 Example 4-25 所示.

4.14.0.2. Example 4-25. 使用 uu 模块解码 uu 格式的文件

```
File: uu-example-2.py



import uu

import StringIO



infile = "samples/sample.uue"

outfile = "samples/sample.jpg"



#

# decode



fi = open(infile)

fo = StringIO.StringIO()
```

```

```

```
uu.decode(fi, fo)
```

```
#
# compare with original data file

data = open(outfile, "rb").read()

if fo.getvalue() == data:
    print len(data), "bytes ok"
```

4.15. binascii 模块

binascii 提供了多个编码的支持函数，包括 base64 ， binhex ， 以及 uu . 如 Example 4-26 所示.

2.0 及以后版本中，你还可以使用它在二进制数据和十六进制字符串中相互转换.

4.15.0.1. Example 4-26. 使用 binascii 模块

```
File: binascii-example-1.py

import binascii
```

```
text = "hello, mrs teal"



data = binascii.b2a_base64(text)

text = binascii.a2b_base64(data)

print text, "<=>", repr(data)



data = binascii.b2a_uu(text)

text = binascii.a2b_uu(data)

print text, "<=>", repr(data)



data = binascii.b2a_hqx(text)

text = binascii.a2b_hqx(data)[0]

print text, "<=>", repr(data)
```

```
# 2.0 and newer
```

```
data = binascii.b2a_hex(text)
```

```
text = binascii.a2b_hex(data)

print text, "<=>", repr(data)
```

```
hello, mrs teal <=> 'aGVsbG8sIG1ycyBOZWFs\012'

hello, mrs teal <=> '/:&5L;&\\L(&UR<R!T96%L\012'

hello, mrs teal <=> 'D\'9XE\'mX)\'ebFb"dC@&X'

hello, mrs teal <=>
'68656c6c6f2c206d7273207465616c'
```

5. 文件格式

5.1. 概览

本章将描述用于处理不同文件格式的模块.

5.1.1. Markup 语言

Python 提供了一些用于处理可扩展标记语言(Extensible Markup Language , XML) 和超文本标记语言(Hypertext Markup Language , HTML)的扩展. Python 同样提供了对标准通用标记语言(Standard Generalized Markup Language , SGML)的支持.

所有这些格式都有着相同的结构, 因为 HTML 和 XML 都来自 SGML . 每个文档都是由起始标签(start tags), 结束标签(end tags), 文本(又叫字符数据), 以及实体引用(entity references)构成:

```
<document name="sample.xml">

    <header>This is a header</header>

    <body>This is the body text.  The text can
contain

    plain text (&quot;character data&quot;), tags,
and

    entities.

    </body>

</document>
```

在这个例子中, <document>, <header>, 以及 <body> 是起始标签. 每个起始标签都有一个对应的结束标签, 使用斜线 "/" 标记. 起始标签可以包含多个属性, 比如这里的 name 属性.

起始标签和它对应的结束标签中的任何东西被称为 *元素(element)*. 这里 document 元素包含 header 和 body 两个元素.

" 是一个字符实体(character entity). 字符实体用于在文本区域中表示特殊的保留字符, 使用 & 指示. 这里它代表一个引号, 常见字符实体还有 " < (<)" 和 " > (>)".

虽然 XML , HTML , SGML 使用相同的结构块, 但它们还有一些不同点. 在 XML 中, 所有元素必须有起始和结束标签, 所有标签必须正确嵌套(well-formed). 而且 XML 是区分大小写的, 所以 <document> 和 <Document> 是不同的元素类型.

HTML 有很高灵活性, HTML 语法分析器一般会自动补全缺失标签; 例如, 当遇到一个以 <P> 标签开始的新段落, 却没有对应结束标签, 语法分析器会自动添加一个 </P> 标签. HTML 也是区分大小写的. 另一方面, XML 允许你定义任何元素, 而 HTML 使用一些由 HTML 规范定义的固定元素.

SGML 有着更高的灵活性, 你可以使用自己的声明(declaration) 定义源文件如何转换到元素结构, DTD (document type description , 文件类型定义)可以用来检查结构并补全缺失标签. 技术上来说, HTML 和 XML 都是 SGML 应用, 有各自的 SGML 声明, 而且 HTML 有一个标准 DTD .

Python 提供了多个 makeup 语言分析器. 由于 SGML 是最灵活的格式, Python 的 sgmllib 事实上很简单. 它不会去处理 DTD , 不过你可以继承它来提供更复杂的功能.

Python 的 HTML 支持基于 SGML 分析器. htmllib 将具体的格式输出工作交给 formatter 对象. formatter 模块包含一些标准格式化标志.

Python 的 XML 支持模块很复杂. 先前是只有与 sgmllib 类似的 xmllib , 后来加入了更高级的 expat 模块(可选). 而最新版本中已经准备废弃 xmllib ,启用 xml 包作为工具集.

5.1.2. 配置文件

ConfigParser 模块用于读取简单的配置文件, 类似 Windows 下的 INI 文件.

netrc 模块用于读取 .netrc 配置文件, shlex 模块用于读取类似 shell 脚本语法的配置文件.

5.1.3. 压缩档案格式

Python 的标准库提供了对 GZIP 和 ZIP (2.0 及以后) 格式的支持. 基于 zlib 模块, gzip 和 zipfile 模块分别用来处理这类文件.

5.2. xmllib 模块

xmllib 已在当前版本中申明不支持.

xmlib 模块提供了一个简单的 XML 语法分析器, 使用正则表达式将 XML 数据分离, 如 Example 5-1 所示. 语法分析器只对文档做基本的检查, 例如是否只有一个顶层元素, 所有的标签是否匹配.

XML 数据一块一块地发送给 xmllib 分析器(例如在网路中传输的数据). 分析器在遇到起始标签, 数据区域, 结束标签, 和实体的时候调用不同的方法.

如果你只是对某些标签感兴趣, 你可以定义特殊的 start_tag 和 end_tag 方法, 这里 tag 是标签名称. 这些 start 函数使用它们对应标签的属性作为参数调用(传递时为一个字典).

5.2.0.1. Example 5-1. 使用 xmllib 模块获取元素的信息

```
File: xmllib-example-1.py
```

```
import xmllib



class Parser(xmllib.XMLParser):

    # get quotation number



    def _ _init_ _(self, file=None):

        xmllib.XMLParser._ _init_ _(self)

        if file:

            self.load(file)



    def load(self, file):

        while 1:

            s = file.read(512)

            if not s:

                break

            self.feed(s)
```

```
            self.close()



        def start_quotation(self, attrs):

            print "id =>", attrs.get("id")

            raise EOFError



    try:

        c = Parser()

        c.load(open("samples/sample.xml"))

    except EOFError:

        pass



    id => 031
```

Example 5-2 展示了一个简单(不完整)的内容输出引擎(rendering engine). 分析器有一个元素堆栈(__tags), 它连同文本片断传递给输出生成器. 生成器会在 style 字典中查询当前标签的层次, 如果不存在, 它将根据样式表创建一个新的样式描述.

5.2.0.2. Example 5-2. 使用 xmllib 模块

```
File: xmllib-example-2.py



import xmllib

import string, sys



STYLESHEET = {
```

```
    # each element can contribute one or more style
elements
```

```
    "quotation": {"style": "italic"},

    "lang": {"weight": "bold"},

    "name": {"weight": "medium"},

}



class Parser(xmllib.XMLParser):

    # a simple styling engine



    def _ _init_ _(self, renderer):
```

```
            xmllib.XMLParser._ _init_ _(self)

            self._ _data = []

            self._ _tags = []

            self._ _renderer = renderer



        def load(self, file):

            while 1:

                s = file.read(8192)

                if not s:

                    break

                self.feed(s)

            self.close()



        def handle_data(self, data):

            self._ _data.append(data)



        def unknown_starttag(self, tag, attrs):
```

```
        if self._ _data:
            text = string.join(self._ _data, "")
            self._ _renderer.text(self._ _tags,
text)
        self._ _tags.append(tag)
        self._ _data = []

    def unknown_endtag(self, tag):
```

```
        self._ _tags.pop()
```

```
        if self._ _data:
            text = string.join(self._ _data, "")
            self._ _renderer.text(self._ _tags,
text)
        self._ _data = []


class DumbRenderer:


    def _ _init_ _(self):
```

```
        self.cache = {}



    def text(self, tags, text):

        # render text in the style given by the tag
stack

        tags = tuple(tags)

        style = self.cache.get(tags)

        if style is None:

            # figure out a combined style

            style = {}

            for tag in tags:

                s = STYLESHEET.get(tag)

                if s:

                    style.update(s)

            self.cache[tags] = style # update cache

        # write to standard output

        sys.stdout.write("%s =>\n" % style)
```

```
        sys.stdout.write("  " + repr(text) + "\n")



#

# try it out



r = DumbRenderer()

c = Parser(r)

c.load(open("samples/sample.xml"))
```

```
{'style': 'italic'} =>
```

```
  'I\'ve had a lot of developers come up to me
and\012say,

  "I haven\'t had this much fun in a long time. It
sure

  beats\012writing '

{'style': 'italic', 'weight': 'bold'} =>

  'Cobol'

{'style': 'italic'} =>
```

```
  '″ __ '

{'style': 'italic', 'weight': 'medium'} =>

  'James Gosling'

{'style': 'italic'} =>

  ', on\012'

{'weight': 'bold'} =>

  'Java'

{'style': 'italic'} =>

  '.'
```

5.3. xml.parsers.expat 模块

(可选) xml.parsers.expat 模块是 James Clark's Expat XML parser 的接口. Example 5-3 展示了这个功能完整且性能很好的语法分析器.

5.3.0.1. Example 5-3. 使用 xml.parsers.expat 模块

```
File: xml-parsers-expat-example-1.py



from xml.parsers import expat
```

```
class Parser:



    def _ _init_ _(self):

        self._parser = expat.ParserCreate()

        self._parser.StartElementHandler =
self.start

        self._parser.EndElementHandler = self.end

        self._parser.CharacterDataHandler =
self.data
```

```
    def feed(self, data):
```

```
        self._parser.Parse(data, 0)



    def close(self):

        self._parser.Parse("", 1) # end of data

        del self._parser # get rid of circular
references



    def start(self, tag, attrs):
```

```
            print "START", repr(tag), attrs



        def end(self, tag):

            print "END", repr(tag)



        def data(self, data):

            print "DATA", repr(data)



    p = Parser()
```

```
    p.feed("<tag>data</tag>")
```

```
    p.close()
```

```
    START u'tag' {}

    DATA u'data'

    END u'tag'
```

注意即使你传入的是普通的文本, 这里的分析器仍然会返回 Unicode 字符串. 默认情况下, 分析器将源文本作为 UTF-8 解析. 如果要使用其他编码, 请确保 XML 文件包含 *encoding* 说明. 如 Example 5-4 所示.

5.3.0.2. Example 5-4. 使用 xml.parsers.expat 模块读取 ISO Latin-1 文本

```
File: xml-parsers-expat-example-2.py



from xml.parsers import expat



class Parser:



    def _ _init_ _(self):

        self._parser = expat.ParserCreate()
```

```
        self._parser.StartElementHandler =
self.start
```

```
        self._parser.EndElementHandler = self.end

        self._parser.CharacterDataHandler =
self.data



    def feed(self, data):

        self._parser.Parse(data, 0)
```

```
    def close(self):

        self._parser.Parse("", 1) # end of data

        del self._parser # get rid of circular
references



    def start(self, tag, attrs):

        print "START", repr(tag), attrs



    def end(self, tag):

        print "END", repr(tag)
```

```

```

```
    def data(self, data):

        print "DATA", repr(data)



p = Parser()

p.feed("""\

<?xml version='1.0' encoding='iso-8859-1'?>
```

```
<author>

<name>fredrik lundh</name>

<city>linköping</city>

</author>

"""

)

p.close()



START u'author' {}

DATA u'\012'
```

```
START u'name' {}
```

```
DATA u'fredrik lundh'

END u'name'

DATA u'\012'

START u'city' {}

DATA u'link\366ping'

END u'city'
```

```
DATA u'\012'

END u'author'
```

5.4. sgmllib 模块

sgmllib 模块，提供了一个基本的 SGML 语法分析器. 它与 xmllib 分析器基本相同，但限制更少(而且不是很完善). 如 Example 5-5 所示.

和在 xmllib 中一样，这个分析器在遇到起始标签，数据区域，结束标签以及实体时调用内部方法. 如果你只是对某些标签感兴趣，那么你可以定义特殊的方法.

5.4.0.1. Example 5-5. 使用 sgmllib 模块提取 Title 元素

```
File: sgmllib-example-1.py
```

```
import sgmllib
```

```
import string



class FoundTitle(Exception):

    pass



class ExtractTitle(sgmllib.SGMLParser):
```

```
    def _ _init_ _(self, verbose=0):
        sgmllib.SGMLParser._ _init_ _(self,
verbose)
        self.title = self.data = None

    def handle_data(self, data):
        if self.data is not None:
            self.data.append(data)

    def start_title(self, attrs):
        self.data = []

    def end_title(self):
        self.title = string.join(self.data, "")
        raise FoundTitle # abort parsing!
```

```
def extract(file):
    # extract title from an HTML/SGML stream
    p = ExtractTitle()
    try:
        while 1:
            # read small chunks
            s = file.read(512)
            if not s:
                break
            p.feed(s)
        p.close()
```

```
    except FoundTitle:
```

```
        return p.title
    return None


#
# try it out
```

```
print "html", "=>",
extract(open("samples/sample.htm"))

print "sgml", "=>",
extract(open("samples/sample.sgm"))
```

```
html => A Title.

sgml => Quotations
```

重载 unknown_starttag 和 unknown_endtag 方法就可以处理所有的标签. 如 Example 5-6 所示.

5.4.0.2. Example 5-6. 使用 sgmllib 模块格式化 SGML 文档

```
File: sgmllib-example-2.py
```

```

```

```
import sgmllib

import cgi, sys



class PrettyPrinter(sgmllib.SGMLParser):

    # A simple SGML pretty printer
```

```
    def _ _init_ _(self):

        # initialize base class

        sgmllib.SGMLParser._ _init_ _(self)

        self.flag = 0



    def newline(self):

        # force newline, if necessary

        if self.flag:

            sys.stdout.write("\n")

        self.flag = 0
```

```
```

```
    def unknown_starttag(self, tag, attrs):

        # called for each start tag



        # the attrs argument is a list of (attr,
value)
```

```
            # tuples.  convert it to a string.

            text = ""

            for attr, value in attrs:

                text = text + " %s='%s'" % (attr,
    cgi.escape(value))



            self.newline()

            sys.stdout.write("<%s%s>\n" % (tag, text))



        def handle_data(self, text):

            # called for each text section

            sys.stdout.write(text)

            self.flag = (text[-1:] != "\n")
```

```
```

```
        def handle_entityref(self, text):

            # called for each entity

            sys.stdout.write("&%s;" % text)
```

```
    def unknown_endtag(self, tag):
        # called for each end tag
        self.newline()
        sys.stdout.write("</%s>" % tag)

#
# try it out

file = open("samples/sample.sgm")

p = PrettyPrinter()
p.feed(file.read())
p.close()
```

```
<chapter>
<title>
```

```
Quotations

<title>

<epigraph>

<attribution>

eff-bot, June 1997

<attribution>

<para>

<quote>

Nobody expects the Spanish Inquisition! Amongst

our weaponry are such diverse elements as fear,
surprise,

ruthless efficiency, and an almost fanatical
devotion to

Guido, and nice red uniforms &mdash; oh, damn!

<quote>
```

```
<para>
```

```
<epigraph>

<chapter>
```

Example 5-7 检查 SGML 文档是否是如 XML 那样 "正确格式化", 所有的元素是否正确嵌套, 起始和结束标签是否匹配等.

我们使用列表保存所有起始标签, 然后检查每个结束标签是否匹配前个起始标签. 最后确认到达文件末尾时没有未关闭的标签.

5.4.0.3. Example 5-7. 使用 sgmllib 模块检查格式

```
File: sgmllib-example-3.py



import sgmllib



class WellFormednessChecker(sgmllib.SGMLParser):

    # check that an SGML document is 'well-formed'

    # (in the XML sense).



    def _ _init_ _(self, file=None):

        sgmllib.SGMLParser._ _init_ _(self)

        self.tags = []
```

```
        if file:
```

```
            self.load(file)
```

```
    def load(self, file):

        while 1:

            s = file.read(8192)

            if not s:

                break

            self.feed(s)

        self.close()



    def close(self):

        sgmllib.SGMLParser.close(self)

        if self.tags:

            raise SyntaxError, "start tag %s not
closed" % self.tags[-1]



    def unknown_starttag(self, start, attrs):
```

```
        self.tags.append(start)
```

```

```

```
    def unknown_endtag(self, end):

        start = self.tags.pop()

        if end != start:

            raise SyntaxError, "end tag %s does't
match start tag %s" %\

                  (end, start)



try:

    c = WellFormednessChecker()

    c.load(open("samples/sample.htm"))

except SyntaxError:

    raise # report error

else:

    print "document is well-formed"



Traceback (innermost last):

...
```

```
SyntaxError: end tag head does't match start tag meta
```

最后，Example 5-8 中的类可以用来过滤 HTML 和 SGML 文档. 继承这个类，然后实现 `start` 和 `end` 方法即可.

5.4.0.4. Example 5-8. 使用 sgmllib 模块过滤 SGML 文档

```
File: sgmllib-example-4.py



import sgmllib

import cgi, string, sys



class SGMLFilter(sgmllib.SGMLParser):

    # sgml filter.  override start/end to
manipulate

    # document elements



    def _ _init_ _(self, outfile=None,
infile=None):

        sgmllib.SGMLParser._ _init_ _(self)
```

```
        if not outfile:
```

```
            outfile = sys.stdout
        self.write = outfile.write
        if infile:
            self.load(infile)


    def load(self, file):
        while 1:
            s = file.read(8192)
            if not s:
                break
            self.feed(s)
        self.close()


    def handle_entityref(self, name):
        self.write("&%s;" % name)


    def handle_data(self, data):
```

```
            self.write(cgi.escape(data))



        def unknown_starttag(self, tag, attrs):

            tag, attrs = self.start(tag, attrs)

            if tag:

                if not attrs:

                    self.write("<%s>" % tag)

                else:

                    self.write("<%s" % tag)

                    for k, v in attrs:

                        self.write(" %s=%s" % (k,
repr(v)))

                    self.write(">")



        def unknown_endtag(self, tag):

            tag = self.end(tag)

            if tag:
```

```
            self.write("</%s>" % tag)



    def start(self, tag, attrs):

        return tag, attrs # override



    def end(self, tag):

        return tag # override



class Filter(SGMLFilter):



    def fixtag(self, tag):

        if tag == "em":

            tag = "i"

        if tag == "string":

            tag = "b"

        return string.upper(tag)
```

```
        def start(self, tag, attrs):

            return self.fixtag(tag), attrs



        def end(self, tag):

            return self.fixtag(tag)



    c = Filter()

    c.load(open("samples/sample.htm"))
```

5.5. htmllib 模块

htmlib 模块包含了一个标签驱动的(tag-driven) HTML 语法分析器, 它会将数据发送至一个格式化对象. 如 Example 5-9 所示. 更多关于如何解析 HTML 的例子请参阅 formatter 模块.

5.5.0.1. Example 5-9. 使用 htmllib 模块

```
    File: htmllib-example-1.py



    import htmllib

    import formatter
```

```
import string



class Parser(htmllib.HTMLParser):

    # return a dictionary mapping anchor texts to
lists

    # of associated hyperlinks



    def _ _init_ _(self, verbose=0):

        self.anchors = {}

        f = formatter.NullFormatter()

        htmllib.HTMLParser._ _init_ _(self, f,
verbose)



    def anchor_bgn(self, href, name, type):

        self.save_bgn()

        self.anchor = href



    def anchor_end(self):
```

```
            text = string.strip(self.save_end())
```

```
            if self.anchor and text:
```

```
                self.anchors[text] =
    self.anchors.get(text, []) + [self.anchor]
```

```
    file = open("samples/sample.htm")

    html = file.read()

    file.close()



    p = Parser()

    p.feed(html)

    p.close()



    for k, v in p.anchors.items():

        print k, "=>", v



    print
```

```
link => ['http://www.python.org']
```

如果你只是想解析一个 HTML 文件, 而不是将它交给输出设备, 那么 sgmllib 模块会是更好的选择.

5.6. htmlentitydefs 模块

htmlentitydefs 模块包含一个由 HTML 中 ISO Latin-1 字符实体构成的字典. 如 Example 5-10 所示.

5.6.0.1. Example 5-10. 使用 htmlentitydefs 模块

```
File: htmlentitydefs-example-1.py



import htmlentitydefs



entities = htmlentitydefs.entitydefs



for entity in "amp", "quot", "copy", "yen":

    print entity, "=", entities[entity]



amp = &
```

```
quot = "

copy = \302\251
```

```
yen = \302\245
```

Example 5-11 展示了如何将正则表达式与这个字典结合起来翻译字符串中的实体（ cgi.escape 的逆向操作).

5.6.0.2. Example 5-11. 使用 htmlentitydefs 模块翻译实体

```
File: htmlentitydefs-example-2.py



import htmlentitydefs

import re

import cgi



pattern = re.compile("&(\w+?);")



def descape_entity(m,
defs=htmlentitydefs.entitydefs):

    # callback: translate one entity to its ISO Latin
value
```

```
        try:
            return defs[m.group(1)]
        except KeyError:
            return m.group(0) # use as is

def descape(string):
    return pattern.sub(descape_entity, string)

print descape("&lt;spam&amp;eggs&gt;")
print descape(cgi.escape("<spam&eggs>"))
```

```
<spam&eggs>
<spam&eggs>
```

最后, Example 5-12 展示了如何将 XML 保留字符和 ISO Latin-1 字符转换为 XML 字符串. 与 `cgi.escape` 相似, 但它会替换非 ASCII 字符.

5.6.0.3. Example 5-12. 转义 ISO Latin-1 实体

```
File: htmlentitydefs-example-3.py
```

```
import htmlentitydefs
```

```
import re, string
```

```
# this pattern matches substrings of reserved and
non-ASCII characters

pattern = re.compile(r"[&<>\"\x80-\xff]+")



# create character map

entity_map = {}



for i in range(256):

    entity_map[chr(i)] = "&%d;" % i



for entity, char in
htmlentitydefs.entitydefs.items():

    if entity_map.has_key(char):

        entity_map[char] = "&%s;" % entity
```

```
def escape_entity(m, get=entity_map.get):

    return string.join(map(get, m.group()), "")
```

```
def escape(string):

    return pattern.sub(escape_entity, string)



print escape("<spam&eggs>")

print escape("\303\245 i \303\245a \303\244 e
\303\266")



&lt;spam&amp;eggs&gt;

&aring; i &aring;a &auml; e &ouml;
```

5.7. formatter 模块

formatter 模块提供了一些可用于 htmllib 的格式类(formatter classes).

这些类有两种, *formatter* 和 *writer*. formatter 将 HTML 解析器的标签和数据流转换为适合输出设备的事件流(event stream), 而 writer 将事件流输出到设备上. 如 Example 5-13 所示.

大多情况下, 你可以使用 *AbstractFormatter* 类进行格式化. 它会根据不同的格式化事件调用 writer 对象的方法. *AbstractWriter* 类在每次方法调用时打印一条信息.

5.7.0.1. Example 5-13. 使用 formatter 模块将 HTML 转换为事件流

```
File: formatter-example-1.py



import formatter

import htmllib



w = formatter.AbstractWriter()

f = formatter.AbstractFormatter(w)



file = open("samples/sample.htm")



p = htmllib.HTMLParser(f)

p.feed(file.read())

p.close()
```

```
    file.close()



    send_paragraph(1)
```

```
    new_font(('h1', 0, 1, 0))
```

```
    send_flowing_data('A Chapter.')

    send_line_break()

    send_paragraph(1)

    new_font(None)

    send_flowing_data('Some text. Some more text.
    Some')

    send_flowing_data(' ')

    new_font((None, 1, None, None))

    send_flowing_data('emphasized')

    new_font(None)

    send_flowing_data(' text. A')

    send_flowing_data(' link')

    send_flowing_data('[1]')
```

```
send_flowing_data('.')
```

formatter 模块还提供了 *NullWriter* 类，它会将任何传递给它的事件忽略；以及 *DumbWriter* 类，它会将事件流转换为纯文本文档. 如 Example 5-14 所示.

5.7.0.2. Example 5-14. 使用 formatter 模块将 HTML 转换为纯文本

```
File: formatter-example-2.py



import formatter

import htmllib



w = formatter.DumbWriter() # plain text

f = formatter.AbstractFormatter(w)



file = open("samples/sample.htm")



# print html body as plain text

p = htmllib.HTMLParser(f)

p.feed(file.read())
```

```
p.close()



file.close()
```

```
# print links
```

```
print

print

i = 1

for link in p.anchorlist:

    print i, "=>", link

    i = i + 1
```

A Chapter.

Some text. Some more text. Some emphasized text. A link[1].

```
1 => http://www.python.org
```

Example 5-15 提供了一个自定义的 Writer ， 它继承自 *DumbWriter* 类, 会记录当前字体样式并根据字体美化输出格式.

5.7.0.3. Example 5-15. 使用 formatter 模块自定义 Writer

```
File: formatter-example-3.py
```

```

```

```
import formatter

import htmllib, string



class Writer(formatter.DumbWriter):



    def _ _init_ _(self):

        formatter.DumbWriter._ _init_ _(self)

        self.tag = ""

        self.bold = self.italic = 0

        self.fonts = []
```

```
    def new_font(self, font):
        if font is None:
            font = self.fonts.pop()
            self.tag, self.bold, self.italic = font
        else:
```

```
            self.fonts.append((self.tag,
self.bold, self.italic))
```

```
            tag, bold, italic, typewriter = font
            if tag is not None:
                self.tag = tag
            if bold is not None:
                self.bold = bold
            if italic is not None:
                self.italic = italic


    def send_flowing_data(self, data):
        if not data:
```

```
            return

        atbreak = self.atbreak or data[0] in
string.whitespace

        for word in string.split(data):

            if atbreak:

                self.file.write(" ")

            if self.tag in ("h1", "h2", "h3"):

                word = string.upper(word)

            if self.bold:

                word = "*" + word + "*"

            if self.italic:

                word = "_" + word + "_"

            self.file.write(word)

            atbreak = 1

        self.atbreak = data[-1] in
string.whitespace



w = Writer()
```

```
f = formatter.AbstractFormatter(w)



file = open("samples/sample.htm")



# print html body as plain text

p = htmllib.HTMLParser(f)

p.feed(file.read())
```

```
p.close()
```

```
_A_ _CHAPTER._



Some text. Some more text. Some *emphasized* text.
A link[1].
```

5.8. ConfigParser 模块

ConfigParser 模块用于读取配置文件.

配置文件的格式与 Windows INI 文件类似, 可以包含一个或多个区域 (section), 每个区域可以有多个配置条目.

这里有个样例配置文件, 在 Example 5-16 用到了这个文件:

```
[book]

title: The Python Standard Library

author: Fredrik Lundh

email: fredrik@pythonware.com

version: 2.0-001115



[ematter]
```

```
pages: 250
```

```


[hardcopy]

pages: 350
```

Example 5-16 使用 ConfigParser 模块读取这个配制文件.

5.8.0.1. Example 5-16. 使用 ConfigParser 模块

```
File: configparser-example-1.py



import ConfigParser
```

```
import string



config = ConfigParser.ConfigParser()



config.read("samples/sample.ini")



# print summary

print
```

```
print string.upper(config.get("book", "title"))
```

```
print "by", config.get("book", "author"),

print  "(" + config.get("book", "email") + ")"

print

print config.get("ematter", "pages"), "pages"

print



# dump entire config file

for section in config.sections():
```

```
    print section

    for option in config.options(section):

        print " ", option, "=", config.get(section,
option)
```

```
THE PYTHON STANDARD LIBRARY

by Fredrik Lundh (fredrik@pythonware.com)



250 pages
```

```

```

```
book

  title = The Python Standard Library

  email = fredrik@pythonware.com

  author = Fredrik Lundh

  version = 2.0-001115

  _ _name_ _ = book

ematter
```

```
    _ _name_ _ = ematter

    pages = 250

  hardcopy

    _ _name_ _ = hardcopy

    pages = 350
```

Python 2.0 以后, ConfigParser 模块也可以将配置数据写入文件, 如 Example 5-17 所示.

5.8.0.2. Example 5-17. 使用 ConfigParser 模块写入配置数据

```
File: configparser-example-2.py
```

```

```

```
import ConfigParser

import sys



config = ConfigParser.ConfigParser()



# set a number of parameters

config.add_section("book")
```

```
config.set("book", "title", "the python standard
library")

config.set("book", "author", "fredrik lundh")



config.add_section("ematter")

config.set("ematter", "pages", 250)



# write to screen

config.write(sys.stdout)
```

```
[book]
```

```
title = the python standard library

author = fredrik lundh



[ematter]

pages = 250
```

5.9. netrc 模块

netrc 模块可以用来解析 *.netrc* 配置文件, 如 Example 5-18 所示. 该文件用于在用户的 home 目录储存 FTP 用户名和密码. (别忘记设置这个文件的属性为: "chmod 0600 ~/.netrc," 这样只有当前用户能访问).

5.9.0.1. Example 5-18. 使用 netrc 模块

```
File: netrc-example-1.py



import netrc







# default is $HOME/.netrc
```

```
info = netrc.netrc("samples/sample.netrc")
```

```



login, account, password =
info.authenticators("secret.fbi")

print "login", "=>", repr(login)

print "account", "=>", repr(account)

print "password", "=>", repr(password)
```

```
login => 'mulder'

account => None

password => 'trustno1'
```

5.10. shlex 模块

shlex 模块为基于 Unix shell 语法的语言提供了一个简单的 lexer (也就是 tokenizer). 如 Example 5-19 所示.

5.10.0.1. Example 5-19. 使用 shlex 模块

```
File: shlex-example-1.py
```

```
import shlex
```

```


lexer = shlex.shlex(open("samples/sample.netrc",
"r"))

lexer.wordchars = lexer.wordchars + ".-"



while 1:

    token = lexer.get_token()
```

```
    if not token:
        break
    print repr(token)
```

```
'machine'
'secret.fbi'
'login'
'mulder'
'password'
'trustno1'
```

```
'machine'
```

```
'non.secret.fbi'
'login'
'scully'
'password'
'noway'
```

5.11. zipfile 模块

(2.0 新增) zipfile 模块可以用来读写 ZIP 格式.

5.11.1. 列出内容

使用 namelist 和 infolist 方法可以列出压缩档的内容, 前者返回由文件名组成的列表, 后者返回由 *ZipInfo* 实例组成的列表. 如 Example 5-20 所示.

5.11.1.1. Example 5-20. 使用 zipfile 模块列出 ZIP 文档中的文件

```
File: zipfile-example-1.py



import zipfile



```

```
file = zipfile.ZipFile("samples/sample.zip", "r")
```

```


# list filenames

for name in file.namelist():

    print name,

print


```

```
# list file information

for info in file.infolist():

    print info.filename, info.date_time,
info.file_size
```

```
sample.txt sample.jpg

sample.txt (1999, 9, 11, 20, 11, 8) 302

sample.jpg (1999, 9, 18, 16, 9, 44) 4762
```

5.11.2. 从 ZIP 文件中读取数据

调用 `read` 方法就可以从 ZIP 文档中读取数据. 它接受一个文件名作为参数, 返回字符串. 如 Example 5-21 所示.

5.11.2.1. Example 5-21. 使用 zipfile 模块从 ZIP 文件中读取数据

```
File: zipfile-example-2.py

import zipfile

file = zipfile.ZipFile("samples/sample.zip", "r")
```

```
for name in file.namelist():

    data = file.read(name)

    print name, len(data), repr(data[:10])



sample.txt 302 'We will pe'

sample.jpg 4762 '\377\330\377\340\000\020JFIF'
```

5.11.3. 向 ZIP 文件写入数据

向压缩档加入文件很简单, 将文件名, 文件在 ZIP 档中的名称传递给 `write` 方法即可.

Example 5-22 将 samples 目录中的所有文件打包为一个 ZIP 文件.

5.11.3.1. Example 5-22. 使用 zipfile 模块将文件储存在 ZIP 文件里

```
File: zipfile-example-3.py



import zipfile

import glob, os



# open the zip file for writing, and write stuff to
it
```

```
file = zipfile.ZipFile("test.zip", "w")



for name in glob.glob("samples/*"):

    file.write(name, os.path.basename(name),
zipfile.ZIP_DEFLATED)



file.close()



# open the file again, to see what's in it



file = zipfile.ZipFile("test.zip", "r")
```

```
for info in file.infolist():
```

```
    print info.filename, info.date_time,
info.file_size, info.compress_size
```

```
sample.wav (1999, 8, 15, 21, 26, 46) 13260 10985

sample.jpg (1999, 9, 18, 16, 9, 44) 4762 4626

sample.au (1999, 7, 18, 20, 57, 34) 1676 1103
```

```
...
```

write 方法的第三个可选参数用于控制是否使用压缩. 默认为 zipfile.ZIP_STORED , 意味着只是将数据储存在档案里而不进行任何压缩. 如果安装了 zlib 模块, 那么就可以使用 zipfile.ZIP_DEFLATED 进行压缩.

zipfile 模块也可以向档案中添加字符串. 不过, 这需要一点技巧, 你需要创建一个 *ZipInfo* 实例, 并正确配置它. Example 5-23 提供了一种简单的解决办法.

5.11.3.2. Example 5-23. 使用 zipfile 模块在 ZIP 文件中储存字符串

```
File: zipfile-example-4.py



import zipfile
```

```
import glob, os, time
```

```


file = zipfile.ZipFile("test.zip", "w")



now = time.localtime(time.time())[:6]



for name in ("life", "of", "brian"):

    info = zipfile.ZipInfo(name)
```

```
    info.date_time = now
    info.compress_type = zipfile.ZIP_DEFLATED
    file.writestr(info, name*1000)

file.close()

# open the file again, to see what's in it

file = zipfile.ZipFile("test.zip", "r")
```

```
for info in file.infolist():
    print info.filename, info.date_time,
info.file_size, info.compress_size
```

```
life (2000, 12, 1, 0, 12, 1) 4000 26
of (2000, 12, 1, 0, 12, 1) 2000 18
brian (2000, 12, 1, 0, 12, 1) 5000 31
```

5.12. gzip 模块

gzip 模块用来读写 gzip 格式的压缩文件, 如 Example 5-24 所示.

5.12.0.1. Example 5-24. 使用 gzip 模块读取压缩文件

```
File: gzip-example-1.py



import gzip



file = gzip.GzipFile("samples/sample.gz")
```

```

```

```
print file.read()



Well it certainly looks as though we're in for

a splendid afternoon's sport in this the 127th

Upperclass Twit of the Year Show.
```

标准的实现并不支持 seek 和 tell 方法. 不过 Example 5-25 可以解决这个问题.

5.12.0.2. Example 5-25. 给 gzip 模块添加 seek/tell 支持

```
File: gzip-example-2.py



import gzip



class gzipFile(gzip.GzipFile):

    # adds seek/tell support to GzipFile



    offset = 0
```

```

```

```
    def read(self, size=None):

        data = gzip.GzipFile.read(self, size)

        self.offset = self.offset + len(data)

        return data



    def seek(self, offset, whence=0):

        # figure out new position (we can only seek
forwards)
```

```
        if whence == 0:
            position = offset
        elif whence == 1:
            position = self.offset + offset
        else:
            raise IOError, "Illegal argument"
        if position < self.offset:
            raise IOError, "Cannot seek backwards"
```

```
        # skip forward, in 16k blocks
```

```
        while position > self.offset:
            if not self.read(min(position -
self.offset, 16384)):
                break

    def tell(self):
        return self.offset
```

```
#

# try it



file = gzipFile("samples/sample.gz")

file.seek(80)



print file.read()



this the 127th
```

```
Upperclass Twit of the Year Show.
```

6. 邮件和新闻消息处理

"To be removed from our list of future commercial postings by [SOME] PUBLISHING COMPANY an Annual Charge of Ninety Five dollars is required. Just send $95.00 with your Name, Address and Name of the Newsgroup to be removed from our list."
- Newsgroup spammer, July 1996
"想要退出 '某' 宣传公司的未来商业广告列表吗, 您需要付 95 美元. 只要您支付 95 美元, 并且告诉我们您的姓名, 地址, 和需要退出的新闻组, 我们就会把您从列表中移除."

- 新闻组垃圾发送者, 1996 年 7 月

6.1. 概览

Python 有大量用于处理邮件和新闻组的模块, 其中包括了许多常见的邮件格式.

6.2. rfc822 模块

rfc822 模块包括了一个邮件和新闻组的解析器 (也可用于其它符合 RFC 822 标准的消息, 比如 HTTP 头).

通常, RFC 822 格式的消息包含一些标头字段, 后面至少有一个空行, 然后是信息主体.

For example, here's a short mail message. The first five lines make up the message header, and the actual message (a single line, in this case) follows after an empty line:

例如这里的邮件信息. 前五行组成了消息标头, 隔一个空行后是消息主体.

```
Message-Id: <20001114144603.00abb310@oreilly.com>
```

```
Date: Tue, 14 Nov 2000 14:55:07 -0500
```

```
To: "Fredrik Lundh" <fredrik@effbot.org>

From: Frank

Subject: Re: python library book!



Where is it?
```

消息解析器读取标头字段后会返回一个以消息标头为键的类字典对象，如 Example 6-1 所示.

6.2.0.1. Example 6-1. 使用 rfc822 模块

```
File: rfc822-example-1.py



import rfc822



file = open("samples/sample.eml")



message = rfc822.Message(file)



for k, v in message.items():
    print k, "=", v


print len(file.read()), "bytes in body"



subject = Re: python library book!
from = "Frank" <your@editor>
```

```
message-id=<20001114144603.00abb310@oreilly.com>

to = "Fredrik Lundh" <fredrik@effbot.org>

date = Tue, 14 Nov 2000 14:55:07 -0500

25 bytes in body
```

消息对象(message object)还提供了一些用于解析地址字段和数据的, 如 Example 6-2 所示.

6.2.0.2. Example 6-2. 使用 rfc822 模块解析标头字段

```
File: rfc822-example-2.py



import rfc822



file = open("samples/sample.eml")
```

```

```

```
message = rfc822.Message(file)



print message.getdate("date")

print message.getaddr("from")
```

```
print message.getaddrlist("to")



(2000, 11, 14, 14, 55, 7, 0, 0, 0)

('Frank', 'your@editor')

[('Fredrik Lundh', 'fredrik@effbot.org')]
```

地址字段被解析为（实际名称，邮件地址）这样的元组. 数据字段被解析为 9 元时间元组，可以使用 time 模块处理.

6.3. mimetools 模块

多用途因特网邮件扩展（Multipurpose Internet Mail Extensions, MIME）标准定义了如何在 RFC 822 格式的消息中储存非 ASCII 文本，图像以及其它数据.

mimetools 模块包含一些读写 MIME 信息的工具. 它还提供了一个类似 rfc822 模块中 *Message* 的类，用于处理 MIME 编码的信息. 如 Example 6-3 所示.

6.3.0.1. Example 6-3. 使用 mimetools 模块

```
File: mimetools-example-1.py



import mimetools



file = open("samples/sample.msg")
```

```
msg = mimetools.Message(file)



print "type", "=>", msg.gettype()

print "encoding", "=>", msg.getencoding()

print "plist", "=>", msg.getplist()



print "header", "=>"

for k, v in msg.items():

    print "  ", k, "=", v
```

```
type => text/plain
```

```
encoding => 7bit

plist => ['charset="iso-8859-1"']

header =>

   mime-version = 1.0

   content-type = text/plain;
```

```
 charset="iso-8859-1"

   to = effbot@spam.egg

   date = Fri, 15 Oct 1999 03:21:15 -0400

   content-transfer-encoding = 7bit

   from = "Fredrik Lundh" <fredrik@pythonware.com>

   subject = By the way...

...
```

6.4. MimeWriter 模块

MimeWriter 模块用于生成符合 MIME 邮件标准的 "多部分" 的信息, 如 Example 6-4 所示.

6.4.0.1. Example 6-4. 使用 MimeWriter 模块

```
File: mimewriter-example-1.py



import MimeWriter



# data encoders

# 数据编码
```

```
import quopri

import base64

import StringIO



import sys



TEXT = """

here comes the image you asked for.  hope

it's what you expected.



</F>"""
```

```

```

```
FILE = "samples/sample.jpg"



file = sys.stdout



#
```

```
# create a mime multipart writer instance



mime = MimeWriter.MimeWriter(file)

mime.addheader("Mime-Version", "1.0")



mime.startmultipartbody("mixed")



# add a text message

# 加入文字信息



part = mime.nextpart()
```

```
part.addheader("Content-Transfer-Encoding",
"quoted-printable")
```

```
part.startbody("text/plain")



quopri.encode(StringIO.StringIO(TEXT), file, 0)
```

```
# add an image

# 加入图片



part = mime.nextpart()

part.addheader("Content-Transfer-Encoding",
"base64")

part.startbody("image/jpeg")



base64.encode(open(FILE, "rb"), file)



mime.lastpart()
```

输出结果如下:

```
Content-Type: multipart/mixed;

    boundary='host.1.-852461.936831373.130.24813'



--host.1.-852461.936831373.130.24813

Content-Type: text/plain
```

```
Context-Transfer-Encoding: quoted-printable



here comes the image you asked for.  hope

it's what you expected.



</F>



--host.1.-852461.936831373.130.24813

Content-Type: image/jpeg

Context-Transfer-Encoding: base64



/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJC
QgKDBQNDAsLDBkSEw8UHRof
```

```
HBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDA
QkJCQwLDBgNDRgyIRwhMjIy
```

```
...

1e5vLrSYbJnEVpEgjCLx5mPU0qsVK0UaxjdNlS+1U6pfzTR8I
zEhj2HrVG6m8m18xc8cIKSC

tCuFyC746j/Cq2pTia4WztfmKjGBXTCmo6IUpt==
```

```
--host.1.-852461.936831373.130.24813--
```

[Example 6-5 #eg-6-5] 使用辅助类储存每个子部分.

6.4.0.2. Example 6-5. MimeWriter 模块的辅助类

```
File: mimewriter-example-2.py



import MimeWriter

import string, StringIO, sys

import re, quopri, base64



# check if string contains non-ascii characters

must_quote = re.compile("[\177-\377]").search
```

```

```

```
#

# encoders
```

```
def encode_quoted_printable(infile, outfile):

    quopri.encode(infile, outfile, 0)



class Writer:



    def _ _init_ _(self, file=None, blurb=None):

        if file is None:

            file = sys.stdout

        self.file = file

        self.mime = MimeWriter.MimeWriter(file)

        self.mime.addheader("Mime-Version",
"1.0")
```

```
        file =
self.mime.startmultipartbody("mixed")
```

```
        if blurb:

            file.write(blurb)
```

```
    def close(self):
        "End of message"
        self.mime.lastpart()
        self.mime = self.file = None

    def write(self, data, mimetype="text/plain"):
        "Write data from string or file to message"

        # data is either an opened file or a string
        if type(data) is type(""):
            file = StringIO.StringIO(data)
        else:
            file = data
            data = None

        part = self.mime.nextpart()
```

```
        typ, subtyp = string.split(mimetype, "/",
1)



        if typ == "text":



            # text data

            encoding = "quoted-printable"

            encoder = lambda i, o: quopri.encode(i,
o, 0)



            if data and not must_quote(data):

                # copy, don't encode

                encoding = "7bit"

                encoder = None
```

```
        else:
```

```
```

```
            # binary data (image, audio,
application, ...)

            encoding = "base64"

            encoder = base64.encode



        #

        # write part headers



        if encoding:

part.addheader("Content-Transfer-Encoding",
encoding)



        part.startbody(mimetype)



        #

        # write part body
```

```

```

```
            if encoder:
```

```
            encoder(file, self.file)

        elif data:

            self.file.write(data)

        else:

            while 1:

                data = infile.read(16384)

                if not data:

                    break

                outfile.write(data)



#

# try it out



BLURB = "if you can read this, your mailer is not
MIME-aware\n"



mime = Writer(sys.stdout, BLURB)
```

```
# add a text message

mime.write("""\

here comes the image you asked for.  hope

it's what you expected.

""", "text/plain")



# add an image

mime.write(open("samples/sample.jpg", "rb"),
"image/jpeg")



mime.close()
```

6.5. mailbox 模块

mailbox 模块用来处理各种不同类型的邮箱格式, 如 Example 6-6 所示. 大部分邮箱格式使用文本文件储存纯 RFC 822 信息, 用分割行区别不同的信息.

6.5.0.1. Example 6-6. 使用 mailbox 模块

```
File: mailbox-example-1.py
```

```
import mailbox



mb =
mailbox.UnixMailbox(open("/var/spool/mail/effbot"
))



while 1:

    msg = mb.next()

    if not msg:

        break

    for k, v in msg.items():

        print k, "=", v

    body = msg.fp.read()

    print len(body), "bytes in body"



subject = for he's a ...

message-id = <199910150027.CAA03202@spam.egg>
```

```
received = (from fredrik@pythonware.com)

 by spam.egg (8.8.7/8.8.5) id CAA03202

 for effbot; Fri, 15 Oct 1999 02:27:36 +0200

from = Fredrik Lundh <fredrik@pythonware.com>

date = Fri, 15 Oct 1999 12:35:36 +0200

to = effbot@spam.egg

1295 bytes in body
```

6.6. mailcap 模块

mailcap 模块用于处理 *mailcap* 文件, 该文件指定了不同的文档格式的处理方法(Unix 系统下). 如 Example 6-7 所示.

6.6.0.1. Example 6-7. 使用 mailcap 模块获得 Capability 字典

```
File: mailcap-example-1.py



import mailcap



caps = mailcap.getcaps()
```

```
for k, v in caps.items():
    print k, "=", v


image/* = [{'view': 'pilview'}]
application/postscript = [{'view': 'ghostview'}]
```

Example 6-7 中, 系统使用 pilview 来预览(view)所有类型的图片, 使用 ghostscript viewer 预览 PostScript 文档. Example 6-8 展示了如何使用 mailcap 获得特定操作的命令.

6.6.0.2. Example 6-8. 使用 mailcap 模块获得打开

```
File: mailcap-example-2.py

import mailcap

caps = mailcap.getcaps()

command, info = mailcap.findmatch(
```

```
    caps, "image/jpeg", "view",
"samples/sample.jpg"

    )



print command



pilview samples/sample.jpg
```

6.7. mimetypes 模块

mimetypes 模块可以判断给定 url（uniform resource locator，统一资源定位符）的 MIME 类型. 它基于一个内建的表, 还可能搜索 Apache 和 Netscape 的配置文件. 如 Example 6-9 所示.

6.7.0.1. Example 6-9. 使用 mimetypes 模块

```
File: mimetypes-example-1.py



import mimetypes

import glob, urllib



for file in glob.glob("samples/*"):
```

```
    url = urllib.pathname2url(file)
    print file, mimetypes.guess_type(url)


samples\sample.au ('audio/basic', None)
samples\sample.ini (None, None)
samples\sample.jpg ('image/jpeg', None)
samples\sample.msg (None, None)
samples\sample.tar ('application/x-tar', None)
samples\sample.tgz ('application/x-tar', 'gzip')
samples\sample.txt ('text/plain', None)
samples\sample.wav ('audio/x-wav', None)
samples\sample.zip ('application/zip', None)
```

6.8. packmail 模块

(已废弃) packmail 模块可以用来创建 Unix shell 档案. 如果安装了合适的工具, 那么你就可以直接通过运行来解开这样的档案. Example 6-10 展示了如何打包单个文件, Example 6-11 则打包了整个目录树.

6.8.0.1. Example 6-10. 使用 packmail 打包单个文件

File: packmail-example-1.py

```
import packmail

import sys



packmail.pack(sys.stdout, "samples/sample.txt",
"sample.txt")



echo sample.txt

sed "s/^X//" >sample.txt <<"!"

XWe will perhaps eventually be writing only small

Xmodules, which are identified by name as they are

Xused to build larger ones, so that devices like

Xindentation, rather than delimiters, might become

Xfeasible for expressing local structure in the

Xsource language.

X    -- Donald E. Knuth, December 1974

!
```

====Example 6-11. 使用 packmail 打包整个目录树===[eg-6-11]

```
File: packmail-example-2.py



import packmail

import sys



packmail.packtree(sys.stdout, "samples")
```

注意, 这个模块不能处理二进制文件, 例如声音或者图像文件.

6.9. mimify 模块

mimify 模块用于在 MIME 编码的文本信息和普通文本信息(例如 ISO Latin 1 文本)间相互转换. 它可以用作命令行工具, 或是特定邮件代理的转换过滤器:

```
$ mimify.py -e raw-message mime-message

$ mimify.py -d mime-message raw-message
```

作为模块使用, 如 Example 6-12 所示.

6.9.0.1. Example 6-12. 使用 mimify 模块解码信息

```
File: mimify-example-1.py
```

```
import mimify

import sys



mimify.unmimify("samples/sample.msg", sys.stdout,
1)
```

这里是一个包含两部分的 MIME 信息, 一个是引用的可打印信息, 另个是 base64 编码信息. unmimify 的第三个参数决定是否自动解码 base64 编码的部分:

```
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary='boundary'



this is a multipart sample file.  the two

parts both contain ISO Latin 1 text, with

different encoding techniques.



--boundary

Content-Type: text/plain
```

```
Content-Transfer-Encoding: quoted-printable
```

```

```

```
sillmj=F6lke! blindstyre! medisterkorv!



--boundary

Content-Type: text/plain

Content-Transfer-Encoding: base64



a29tIG5lciBiYXJhLCBvbSBkdSB09nJzIQ==



--boundary--
```

解码结果如下 (可读性相对来说更好些):

```
MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=
'boundary'



this is a multipart sample file.  the two
```

```
parts both contain ISO Latin 1 text, with
```

```
different encoding techniques.
```

```

--boundary

Content-Type: text/plain



sillmjölke! blindstyre! medisterkorv!



--boundary

Content-Type: text/plain



kom ner bara, om du törs!
```

Example 6-13 展示了如何编码信息.

6.9.0.2. Example 6-13. 使用 mimify 模块编码信息

```
File: mimify-example-2.py
```

```
import mimify

import StringIO, sys
```

```

```

```
#

# decode message into a string buffer



file = StringIO.StringIO()



mimify.unmimify("samples/sample.msg", file, 1)



#

# encode message from string buffer



file.seek(0) # rewind



mimify.mimify(file, sys.stdout)
```

6.10. multifile 模块

multifile 模块允许你将一个多部分的 MIME 信息的每部分作为单独的文件处理. 如 Example 6-14 所示.

6.10.0.1. Example 6-14. 使用 multifile 模块

```
File: multifile-example-1.py



import multifile

import cgi, rfc822



infile = open("samples/sample.msg")



message = rfc822.Message(infile)



# print parsed header

for k, v in message.items():

    print k, "=", v
```

```
# use cgi support function to parse content-type
header
```

```
type, params =
cgi.parse_header(message["content-type"])
```

```
if type[:10] == "multipart/":

    # multipart message

    boundary = params["boundary"]

    file = multifile.MultiFile(infile)

    file.push(boundary)

    while file.next():

        submessage = rfc822.Message(file)
```

```
            # print submessage

            print "-" * 68
```

```
            for k, v in submessage.items():
```

```
                print k, "=", v

            print

            print file.read()



        file.pop()



    else:



        # plain message

        print infile.read()
```

7. 网络协议

"Increasingly, people seem to misinterpret complexity as sophistication, which is baffling - the incomprehensible should cause suspicion rather

than admiration. Possibly this trend results from a mistaken belief that using a somewhat mysterious device confers an aura of power on the user." - Niklaus Wirth

---

7.1. 概览

本章描述了 Python 的 socket 协议支持以及其他建立在 socket 模块上的网络模块. 这些包含了对大多流行 Internet 协议客户端的支持, 以及一些可用来实现 Internet 服务器的框架.

对于那些本章中的底层的例子, 我将使用两个协议作为样例: Internet Time Protocol (Internet 时间协议) 以及 Hypertext Transfer Protocol (超文本传输协议, HTTP 协议).

7.1.1. Internet 时间协议

Internet 时间协议 (RFC 868, Postel 和 Harrenstien, 1983) 可以让一个网络客户端获得一个服务器的当前时间.

因为这个协议是轻量级的, 许多 Unix 系统(但不是所有)都提供了这个服务. 它可能是最简单的网络协议了. 服务器等待连接请求并在连接后返回当前时间 (4 字节整数, 自从 1900 年 1 月 1 日到当前的秒数).

协议很简单, 这里我们提供规格书给大家:

```
File: rfc868.txt




Network Working Group
J. Postel - ISI


Request for Comments: 868
K. Harrenstien - SRI



May 1983
```

Time Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on

the ARPA Internet that choose to implement a Time Protocol are expected

to adopt and implement this standard.

本 RFC 规范提供了一个 ARPA Internet community 上的标准.

在 ARPA Internet 上的所有主机应当采用并实现这个标准.

This protocol provides a site-independent, machine readable date and

time. The Time service sends back to the originating source the time in

seconds since midnight on January first 1900.

此协议提供了一个独立于站点的, 机器可读的日期和时间信息.

时间服务返回的是从 1900 年 1 月 1 日午夜到现在的秒数.

One motivation arises from the fact that not all systems have a

date/time clock, and all are subject to occasional human or machine

error. The use of time-servers makes it possible to quickly confirm or

correct a system's idea of the time, by making a brief poll of several

independent sites on the network.

设计这个协议的一个重要目的在于, 网络上的一些主机并没有时钟,

这有可能导致人工或者机器错误. 我们可以依靠时间服务器快速确认或者修改

一个系统的时间.

This protocol may be used either above the Transmission Control Protocol

(TCP) or above the User Datagram Protocol (UDP).

该协议可以用在 TCP 协议或是 UDP 协议上.

When used via TCP the time service works as follows:

通过 TCP 访问时间服务器的步骤:

* S: Listen on port 37 (45 octal).
* U: Connect to port 37.
* S: Send the time as a 32 bit binary number.
* U: Receive the time.
* U: Close the connection.
* S: Close the connection.

* S: 监听 37 (45 的八进制) 端口.
* U: 连接 37 端口.
* S: 将时间作为 32 位二进制数字发送.

```
 *  U: 接收时间.

 *  U: 关闭连接.

 *  S: 关闭连接.
```

```
   The server listens for a connection on port 37.
When the connection
```

```
   is established, the server returns a 32-bit time
value and closes the

   connection.  If the server is unable to determine
the time at its

   site, it should either refuse the connection or
close it without

   sending anything.
```

```
   服务器在 37 端口监听. 当连接建立的时候, 服务器
返回一个 32 位的数字值

   并关闭连接. 如果服务器自己无法决定当前时间, 那
么它应该拒绝这个连接或者

   不发送任何数据立即关闭连接.
```

When used via UDP the time service works as follows:

通过 TCP 访问时间服务器的步骤:

S: Listen on port 37 (45 octal).

U: Send an empty datagram to port 37.

S: Receive the empty datagram.

S: Send a datagram containing the time as a 32 bit binary number.

U: Receive the time datagram.

S: 监听 37 (45 的八进制) 端口.

U: 发送空数据报文到 37 端口.

S: 接受空报文.

S: 发送包含时间(32 位二进制数字)的报文.

U: 接受时间报文.

The server listens for a datagram on port 37. When a datagram

arrives, the server returns a datagram containing the 32-bit time

value. If the server is unable to determine the time at its site, it

should discard the arriving datagram and make no reply.

服务器在 37 端口监听报文. 当报文到达时, 服务器返回包含 32 位时间值

的报文. 如果服务器无法决定当前时间, 那么它应该丢弃到达的报文,

不做任何回复.

The Time

时间

The time is the number of seconds since 00:00 (midnight) 1 January 1900

GMT, such that the time 1 is 12:00:01 am on 1 January 1900 GMT; this

base will serve until the year 2036.

时间是自 1900 年 1 月 1 日 0 时到当前的秒数,

这个协议标准会一直服务到 2036 年. 到时候数字不够用再说.

For example:

   the time 2,208,988,800 corresponds to 00:00 1 Jan 1970 GMT,

             2,398,291,200 corresponds to 00:00 1 Jan 1976 GMT,

             2,524,521,600 corresponds to 00:00 1 Jan 1980 GMT,

             2,629,584,000 corresponds to 00:00 1 May 1983 GMT,

        and -1,297,728,000 corresponds to 00:00 17 Nov 1858 GMT.

例如:

时间值 2,208,988,800 对应 to 00:00 1 Jan 1970 GMT,

2,398,291,200 对应 to 00:00 1 Jan 1976 GMT,

2,524,521,600 对应 to 00:00 1 Jan 1980 GMT,

2,629,584,000 对应 to 00:00 1 May 1983 GMT,

最后 -1,297,728,000 对应 to 00:00 17 Nov 1858 GMT.

RFC868.txt Translated By Andelf(gt: andelf@gmail.com)

7.1.2. HTTP 协议

超文本传输协议（ HTTP, RFC 2616 ）是另个完全不同的东西. 最近的格式说明书(Version 1.1)超过了 100 页.

从它最简单的格式来看，这个协议是很简单的. 客户端发送如下的请求到服务器，请求一个文件:

```
GET /hello.txt HTTP/1.0

Host: hostname

User-Agent: name



[optional request body , 可选的请求正文]
```

服务器返回对应的响应:

```
HTTP/1.0 200 OK

Content-Type: text/plain

Content-Length: 7



Hello
```

请求和响应的 headers (报头)一般会包含更多的域，但是请求 header 中的 Host 域/字段是必须提供的.

header 行使用 "\r\n" 分割，而且 header 后必须有一个空行，即使没有正文 (请求和响应都必须符合这条规则).

剩下的 HTTP 协议格式说明书细节，例如内容协商，缓存机制，保持连接，等等，请参阅 Hypertext TransferProtocol - HTTP/1.1 (http://www.w3.org/Protocols).

7.2. socket 模块

socket 模块实现了到 socket 通讯层的接口. 你可以使用该模块创建客户端或是服务器的 socket .

我们首先以一个客户端为例, Example 7-1 中的客户端连接到一个时间协议服务器, 读取 4 字节的返回数据, 并把它转换为一个时间值.

7.2.0.1. Example 7-1. 使用 socket 模块实现一个时间客户端

```
File: socket-example-1.py



import socket

import struct, time



# server

HOST = "www.python.org"

PORT = 37



# reference time (in seconds since 1900-01-01
00:00:00)

TIME1970 = 2208988800L # 1970-01-01 00:00:00
```

```
# connect to server

s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

s.connect((HOST, PORT))



# read 4 bytes, and convert to time value
```

```
t = s.recv(4)
```

```
t = struct.unpack("!I", t)[0]

t = int(t - TIME1970)



s.close()



# print results

print "server time is", time.ctime(t)

print "local clock is", int(time.time()) - t,
"seconds off"



server time is Sat Oct 09 16:42:36 1999
```

```
local clock is 8 seconds off
```

socket 工厂函数(factory function)根据给定类型(该例子中为 Internet stream socket , 即就是 TCP socket)创建一个新的 socket . connect 方法尝试将这个 socket 连接到指定服务器上. 成功后, 就可以使用 recv 方法读取数据.

创建一个服务器 socket 使用的是相同的方法, 不过这里不是连接到服务器, 而是将 socket bind (绑定)到本机的一个端口上, 告诉它去监听连接请求, 然后尽快处理每个到达的请求.

Example 7-2 创建了一个时间服务器, 绑定到本机的 8037 端口(1024 前的所有端口是为系统服务保留的, Unix 系统下访问它们你必须要有 root 权限).

7.2.0.2. Example 7-2. 使用 socket 模块实现一个时间服务器

```
File: socket-example-2.py



import socket

import struct, time



# user-accessible port

PORT = 8037



# reference time

TIME1970 = 2208988800L


```

```
# establish server

service = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

service.bind(("", PORT))

service.listen(1)
```

```
print "listening on port", PORT



while 1:

    # serve forever

    channel, info = service.accept()

    print "connection from", info

    t = int(time.time()) + TIME1970

    t = struct.pack("!I", t)

    channel.send(t) # send timestamp

    channel.close() # disconnect
```

```
listening on port 8037

connection from ('127.0.0.1', 1469)

connection from ('127.0.0.1', 1470)

...
```

listen 函数的调用告诉 socket 我们期望接受连接. 参数代表连接的队列(用于在程序没有处理前保持连接)大小. 最后 accept 循环将当前时间返回给每个连接的客户端.

注意这里的 accept 函数返回一个新的 socket 对象, 这个对象是直接连接到客户端的. 而原 socket 只是用来保持连接; 所有后来的数据传输操作都使用新的 socket .

我们可以使用 Example 7-3 , (Example 7-1 的通用化版本)来测试这个服务器, .

7.2.0.3. Example 7-3. 一个时间协议客户端

```
File: timeclient.py



import socket

import struct, sys, time



# default server

host = "localhost"

port = 8037
```

```
# reference time (in seconds since 1900-01-01
00:00:00)

TIME1970 = 2208988800L # 1970-01-01 00:00:00
```

```

```

```
def gettime(host, port):

    # fetch time buffer from stream server

    s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

    s.connect((host, port))

    t = s.recv(4)

    s.close()

    t = struct.unpack("!I", t)[0]

    return int(t - TIME1970)



if _ _name_ _ == "_ _main_ _":

    # command-line utility

    if sys.argv[1:]:
```

```
        host = sys.argv[1]

        if sys.argv[2:]:

            port = int(sys.argv[2])

        else:

            port = 37 # default for public servers


    t = gettime(host, port)

    print "server time is", time.ctime(t)

    print "local clock is", int(time.time()) - t,
"seconds off"



server time is Sat Oct 09 16:58:50 1999

local clock is 0 seconds off
```

Example 7-3 所示的脚本也可以作为模块使用；你只需要导入 `timeclient` 模块，然后调用它的 `gettime` 函数.

目前为止，我们已经使用了流（TCP）socket．时间协议还提到了 UDP sockets (报文)．流 socket 的工作模式和电话线类似；你会知道在远端是否有人拿起接听器，在对方挂断的时候你也会注意到．相比之下，发送报文更像是在一间黑屋子里大声喊．可能某人会在那里，但你只有在他回复的时候才会知道.

如 Example 7-4 所示，你不需要在通过报文 socket 发送数据时连接远程机器．只需使用 `sendto` 方法，它接受数据和接收者地址作为参数．读取报文的时候使用 `recvfrom` 方法.

7.2.0.4. Example 7-4. 使用 socket 模块实现一个报文时间客户端

```
File: socket-example-4.py
```

```
import socket
```

```
import struct, time



# server

HOST = "localhost"

PORT = 8037



# reference time (in seconds since 1900-01-01
00:00:00)

TIME1970 = 2208988800L # 1970-01-01 00:00:00



# connect to server

s = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
```

```
# send empty packet

s.sendto("", (HOST, PORT))



# read 4 bytes from server, and convert to time value
```

```
t, server = s.recvfrom(4)
```

```
t = struct.unpack("!I", t)[0]

t = int(t - TIME1970)



s.close()



print "server time is", time.ctime(t)

print "local clock is", int(time.time()) - t,
"seconds off"



server time is Sat Oct 09 16:42:36 1999

local clock is 8 seconds off
```

这里的 recvfrom 返回两个值: 数据和发送者的地址. 后者用于发送回复数据.

Example 7-5 展示了对应的服务器代码.

Example 7-5. 使用 socket 模块实现一个报文时间服务器

```
File: socket-example-5.py



import socket
```

```
import struct, time
```

```


# user-accessible port

PORT = 8037



# reference time

TIME1970 = 2208988800L



# establish server

service = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)

service.bind(("", PORT))


```

```
print "listening on port", PORT



while 1:

    # serve forever

    data, client = service.recvfrom(0)
```

```
    print "connection from", client
```

```
    t = int(time.time()) + TIME1970

    t = struct.pack("!I", t)

    service.sendto(t, client) # send timestamp



listening on port 8037

connection from ('127.0.0.1', 1469)

connection from ('127.0.0.1', 1470)

...
```

最主要的不同在于服务器使用 bind 来分配一个已知端口给 socket ， 根据 recvfrom 函数返回的地址向客户端发送数据.

7.3. select 模块

select 模块允许你检查一个或多个 socket，管道，以及其他流兼容对象所接受的数据，如 Example 7-6 所示.

你可以将一个或更多 socket 传递给 select 函数，然后等待它们状态改变(可读，可写，或是发送错误信号):

- 如果某人在调用了 listen 函数后连接，当远端数据到达时，socket 就成为可读的(这意味着 accept 不会阻塞). 或者是 socket 被关闭或重置时(在此情况下，recv 会返回一个空字符串).
- 当非阻塞调用 connect 方法后建立连接或是数据可以被写入到 socket 时，socket 就成为可写的.
- 当非阻塞调用 connect 方法后连接失败后，socket 会发出一个错误信号.

7.3.0.1. Example 7-6. 使用 select 模块等待经 socket 发送的数据

File: select-example-1.py

```
import select

import socket

import time



PORT = 8037



TIME1970 = 2208988800L
```

```
service = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

```
service.bind(("", PORT))
```

```
service.listen(1)



print "listening on port", PORT



while 1:

    is_readable = [service]

    is_writable = []

    is_error = []

    r, w, e = select.select(is_readable,
is_writable, is_error, 1.0)

    if r:

        channel, info = service.accept()

        print "connection from", info

        t = int(time.time()) + TIME1970

        t = chr(t>>24&255) + chr(t>>16&255) +
chr(t>>8&255) + chr(t&255)
```

```
            channel.send(t) # send timestamp

            channel.close() # disconnect

        else:

            print "still waiting"
```

```
listening on port 8037
still waiting
still waiting
connection from ('127.0.0.1', 1469)
still waiting
connection from ('127.0.0.1', 1470)
...
```

在 Example 7-6 中，我们等待监听 socket 变成可读状态，这代表有一个连接请求到达. 我们用和之前一样的方法处理 channel socket，因为它不可能因为等待 4 字节而填充网络缓冲区. 如果你需要向客户端发送大量的数据，那么你应该在循环的顶端把数据加入到 is_writable 列表中，并且只在 select 允许的情况下写入.

如果你设置 socket 为*非阻塞*模式(通过调用 setblocking 方法)，那么你就可以使用 select 来等待 socket 连接. 不过 asyncore 模块(参见下一节)提供了一个强大的框架，它自动为你处理好了这一切. 所以我不准备在这里多说什么，看下一节吧.

7.4. asyncore 模块

asyncore 模块提供了一个 "反馈性的(reactive)" socket 实现. 该模块允许你定义特定过程完成后所执行的代码, 而不是创建 socket 对象, 调用它们的方法. 你只需要继承 *dispatcher* 类, 然后重载如下方法 (可以选择重载某一个或多个)就可以实现异步的 socket 处理器.

- handle_connect : 一个连接成功建立后被调用.
- handle_expt : 连接失败后被调用.
- handle_accept : 连接请求建立到一个监听 socket 上时被调用. 回调时(callback)应该使用 accept 方法来获得客户端 socket .
- handle_read : 有来自 socket 的数据等待读取时被调用. 回调时应该使用 recv 方法来获得数据.
- handle_write : socket 可以写入数据的时候被调用. 使用 send 方法写入数据.
- handle_close : 当 socket 被关闭或复位时被调用.
- handle_error(type, value, traceback) 在任何一个回调函数发生 Python 错误时被调用. 默认的实现会打印跟踪返回消息到 sys.stdout .

Example 7-7 展示了一个时间客户端, 和 socket 模块中的那个类似.

7.4.0.1. Example 7-7. 使用 asyncore 模块从时间服务器获得时间

```
File: asyncore-example-1.py



import asyncore

import socket, time

```

```

```

```
# reference time (in seconds since 1900-01-01
00:00:00)
```

```
TIME1970 = 2208988800L # 1970-01-01 00:00:00



class TimeRequest(asyncore.dispatcher):

    # time requestor (as defined in RFC 868)



    def _ _init_ _(self, host, port=37):

        asyncore.dispatcher._ _init_ _(self)

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.connect((host, port))



    def writable(self):

        return 0 # don't have anything to write



    def handle_connect(self):

        pass # connection succeeded
```

```
    def handle_expt(self):

        self.close() # connection failed, shutdown



    def handle_read(self):

        # get local time

        here = int(time.time()) + TIME1970



        # get and unpack server time

        s = self.recv(4)

        there = ord(s[3]) + (ord(s[2])<<8) +
(ord(s[1])<<16) + (ord(s[0])<<24L)



        self.adjust_time(int(here - there))



        self.handle_close() # we don't expect more
data



    def handle_close(self):
```

```
            self.close()



        def adjust_time(self, delta):

            # override this method!

            print "time difference is", delta



    #

    # try it out



    request = TimeRequest("www.python.org")



    asyncore.loop()



    log: adding channel <TimeRequest  at 8cbe90>

    time difference is 28

    log: closing channel 192:<TimeRequest connected at
    8cbe90>
```

如果你不想记录任何信息，那么你可以在你的 *dispatcher* 类里重载 `log` 方法.

Example 7-8 展示了对应的时间服务器. 注意这里它使用了两个 *dispatcher* 子类，一个用于监听 socket ，另个用于与客户端通讯.

7.4.0.2. Example 7-8. 使用 asyncore 模块实现时间服务器

```
File: asyncore-example-2.py



import asyncore

import socket, time



# reference time

TIME1970 = 2208988800L



class TimeChannel(asyncore.dispatcher):



    def handle_write(self):

        t = int(time.time()) + TIME1970

        t = chr(t>>24&255) + chr(t>>16&255) +
chr(t>>8&255) + chr(t&255)

        self.send(t)
```

```
        self.close()



class TimeServer(asyncore.dispatcher):



    def _ _init_ _(self, port=37):

        self.port = port

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.bind(("", port))

        self.listen(5)

        print "listening on port", self.port



    def handle_accept(self):

        channel, addr = self.accept()

        TimeChannel(channel)



server = TimeServer(8037)

asyncore.loop()
```

```
log: adding channel <TimeServer  at 8cb940>

listening on port 8037

log: adding channel <TimeChannel  at 8b2fd0>

log: closing channel 52:<TimeChannel connected at
8b2fd0>
```

除了 *dispatcher* 外，这个模块还包含一个 *dispatcher_with_send* 类. 你可以使用这个类发送大量的数据而不会阻塞网络通讯缓冲区.

Example 7-9 中的模块通过继承 *dispatcher_with_send* 类定义了一个 *AsyncHTTP* 类. 当你创建一个它的实例后，它会发出一个 HTTP GET 请求并把接受到的数据发送到一个 "consumer" 目标对象

7.4.0.3. Example 7-9. 使用 asyncore 模块发送 HTTP 请求

```
File: SimpleAsyncHTTP.py



import asyncore

import string, socket

import StringIO

import mimetools, urlparse



class AsyncHTTP(asyncore.dispatcher_with_send):
```

```
    # HTTP requester



    def _ _init_ _(self, uri, consumer):

        asyncore.dispatcher_with_send._ _init_
_(self)



        self.uri = uri

        self.consumer = consumer



        # turn the uri into a valid request

        scheme, host, path, params, query, fragment
= urlparse.urlparse(uri)

        assert scheme == "http", "only supports HTTP
requests"

        try:

            host, port = string.split(host, ":", 1)

            port = int(port)

        except (TypeError, ValueError):

            port = 80 # default port
```

```
        if not path:

            path = "/"

        if params:

            path = path + ";" + params

        if query:

            path = path + "?" + query



        self.request = "GET %s
HTTP/1.0\r\nHost: %s\r\n\r\n" % (path, host)



        self.host = host

        self.port = port



        self.status = None

        self.header = None



        self.data = ""
```

```
        # get things going!

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.connect((host, port))



    def handle_connect(self):

        # connection succeeded

        self.send(self.request)



    def handle_expt(self):

        # connection failed; notify consumer
(status is None)

        self.close()

        try:

            http_header =
self.consumer.http_header

        except AttributeError:

            pass

        else:
```

```
            http_header(self)



    def handle_read(self):

        data = self.recv(2048)

        if not self.header:

            self.data = self.data + data

            try:

                i = string.index(self.data,
"\r\n\r\n")

            except ValueError:

                return # continue

            else:

                # parse header

                fp =
StringIO.StringIO(self.data[:i+4])

                # status line is "HTTP/version
status message"

                status = fp.readline()

                self.status = string.split(status,
" ", 2)
```

```
                # followed by a rfc822-style
message header

                self.header =
mimetools.Message(fp)

                # followed by a newline, and the
payload (if any)

                data = self.data[i+4:]

                self.data = ""

                # notify consumer (status is
non-zero)

                try:

                    http_header =
self.consumer.http_header

                except AttributeError:

                    pass

                else:

                    http_header(self)

                if not self.connected:

                    return # channel was closed by
consumer
```

```
            self.consumer.feed(data)



        def handle_close(self):

            self.consumer.close()

            self.close()
```

Example 7-10 中的小脚本展示了如何使用这个类.

7.4.0.4. Example 7-10. 使用 SimpleAsyncHTTP 类

```
    File: asyncore-example-3.py



    import SimpleAsyncHTTP

    import asyncore



    class DummyConsumer:

        size = 0



        def http_header(self, request):

            # handle header
```

```
        if request.status is None:
            print "connection failed"
        else:
            print "status", "=>", request.status
            for key, value in
request.header.items():
                print key, "=", value


    def feed(self, data):
        # handle incoming data
        self.size = self.size + len(data)


    def close(self):
        # end of data
        print self.size, "bytes in body"


#
# try it out
```

```
consumer = DummyConsumer()



request = SimpleAsyncHTTP.AsyncHTTP(

    "http://www.pythonware.com",

    consumer

    )



asyncore.loop()



log: adding channel <AsyncHTTP  at 8e2850>

status => ['HTTP/1.1', '200', 'OK\015\012']

server = Apache/Unix (Unix)

content-type = text/html

content-length = 3730

...

3730 bytes in body
```

```
log: closing channel 156:<AsyncHTTP connected at
8e2850>
```

这里的 consumer 接口设计时是为了与 htmllib 和 xmllib 分析器兼容的，这样你就可以直接方便地解析 HTML 或是 XML 数据. http_header 方法是可选的；如果没有定义它，那么它将被忽略.

Example 7-10 的一个问题是它不能很好地处理重定向资源. Example 7-11 加入了一个额外的 consumer 层，它可以很好地处理重定向.

7.4.0.5. Example 7-11. 使用 SimpleAsyncHTTP 类处理重定向

```
File: asyncore-example-4.py



import SimpleAsyncHTTP

import asyncore



class DummyConsumer:

    size = 0



    def http_header(self, request):

        # handle header

        if request.status is None:
```

```
            print "connection failed"

        else:

            print "status", "=>", request.status

            for key, value in
request.header.items():

                print key, "=", value



    def feed(self, data):

        # handle incoming data

        self.size = self.size + len(data)



    def close(self):

        # end of data

        print self.size, "bytes in body"



class RedirectingConsumer:



    def _ _init_ _(self, consumer):
```

```
            self.consumer = consumer



        def http_header(self, request):

            # handle header

            if request.status is None or\

               request.status[1] not in ("301", "302"):

                try:

                    http_header =
    self.consumer.http_header

                except AttributeError:

                    pass

                else:

                    return http_header(request)

            else:

                # redirect!

                uri = request.header["location"]

                print "redirecting to", uri, "..."

                request.close()
```

```
            SimpleAsyncHTTP.AsyncHTTP(uri, self)



    def feed(self, data):

        self.consumer.feed(data)



    def close(self):

        self.consumer.close()



#

# try it out



consumer = RedirectingConsumer(DummyConsumer())



request = SimpleAsyncHTTP.AsyncHTTP(

    "http://www.pythonware.com/library",

    consumer

    )
```

```
asyncore.loop()



log: adding channel <AsyncHTTP  at 8e64b0>

redirecting to
http://www.pythonware.com/library/ ...

log: closing channel 48:<AsyncHTTP connected at
8e64b0>

log: adding channel <AsyncHTTP  at 8ea790>

status => ['HTTP/1.1', '200', 'OK\015\012']

server = Apache/Unix (Unix)

content-type = text/html

content-length = 387

...

387 bytes in body

log: closing channel 236:<AsyncHTTP connected at
8ea790>
```

如果服务器返回状态 301 (永久重定向) 或者是 302 (临时重定向), 重定向的 consumer 会关闭当前请求并向新地址发出新请求. 所有对 consumer 的其他调用传递给原来的 consumer .

7.5. asynchat 模块

asynchat 模块是对 asyncore 的一个扩展. 它提供对面向行(line-oriented)的协议的额外支持. 它还提供了增强的缓冲区支持(通过 push 方法和 "producer" 机制.

Example 7-12 实现了一个很小的 HTTP 服务器. 它只是简单地返回包含 HTTP 请求信息的 HTML 文档(浏览器窗口出现的输出).

7.5.0.1. Example 7-12. 使用 asynchat 模块实现一个迷你 HTTP 服务器

```
File: asynchat-example-1.py



import asyncore, asynchat

import os, socket, string



PORT = 8000



class HTTPChannel(asynchat.async_chat):



    def __init__(self, server, sock, addr):

        asynchat.async_chat.__init__(self, sock)

        self.set_terminator("\r\n")
```

```
        self.request = None
        self.data = ""
        self.shutdown = 0


    def collect_incoming_data(self, data):
        self.data = self.data + data


    def found_terminator(self):
        if not self.request:
            # got the request line
            self.request = string.split(self.data,
None, 2)
            if len(self.request) != 3:
                self.shutdown = 1
            else:
                self.push("HTTP/1.0 200 OK\r\n")
                self.push("Content-type:
text/html\r\n")
```

```
                self.push("\r\n")
```

```
            self.data = self.data + "\r\n"
```

```
            self.set_terminator("\r\n\r\n") # look
for end of headers

        else:

            # return payload.

            self.push("<html><body><pre>\r\n")

            self.push(self.data)

            self.push("</pre></body></html>\r\n")

            self.close_when_done()



class HTTPServer(asyncore.dispatcher):



    def _ _init_ _(self, port):

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.bind(("", port))

        self.listen(5)
```

```
    def handle_accept(self):
        conn, addr = self.accept()
        HTTPChannel(self, conn, addr)

#
# try it out

s = HTTPServer(PORT)
print "serving at port", PORT, "..."
asyncore.loop()
```

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en, sv
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; Bruce/1.0)
```

```
Host: localhost:8000

Connection: Keep-Alive
```

producer 接口允许你传入("push")太大以至于无法在内存中储存的对象. asyncore 在需要更多数据的时候自动调用 producer 的 more 方法. 另外, 它使用一个空字符串标记文件的末尾.

Example 7-13 实现了一个很简单的基于文件的 HTTP 服务器, 它使用了一个简单的 *FileProducer* 类来从文件中读取数据, 每次只读取几 kb .

7.5.0.2. Example 7-13. 使用 asynchat 模块实现一个简单的 HTTP 服务器

```
File: asynchat-example-2.py



import asyncore, asynchat

import os, socket, string, sys

import StringIO, mimetools



ROOT = "."



PORT = 8000



class HTTPChannel(asynchat.async_chat):
```

```
    def _ _init_ _(self, server, sock, addr):
        asynchat.async_chat._ _init_ _(self, sock)
        self.server = server
        self.set_terminator("\r\n\r\n")
        self.header = None
        self.data = ""
        self.shutdown = 0

    def collect_incoming_data(self, data):
        self.data = self.data + data
        if len(self.data) > 16384:
            # limit the header size to prevent
attacks
            self.shutdown = 1

    def found_terminator(self):
```

```
        if not self.header:
            # parse http header
            fp = StringIO.StringIO(self.data)
            request = string.split(fp.readline(),
None, 2)
```

```
            if len(request) != 3:
```

```
                # badly formed request; just shut
down
                self.shutdown = 1
            else:
                # parse message header
                self.header =
mimetools.Message(fp)
                self.set_terminator("\r\n")
                self.server.handle_request(
                    self, request[0], request[1],
self.header
                    )
                self.close_when_done()
```

```
            self.data = ""

        else:

            pass # ignore body data, for now



    def pushstatus(self, status,
explanation="OK"):
```

```
        self.push("HTTP/1.0 %d %s\r\n" % (status,
explanation))
```

```
class FileProducer:

    # a producer that reads data from a file object



    def _ _init_ _(self, file):

        self.file = file



    def more(self):

        if self.file:
```

```
            data = self.file.read(2048)

            if data:

                return data

            self.file = None

        return ""
```

```
class HTTPServer(asyncore.dispatcher):



    def _ _init_ _(self, port=None, request=None):

        if not port:

            port = 80

        self.port = port

        if request:

            self.handle_request = request #
external request handler

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)
```

```
            self.bind(("", port))

            self.listen(5)



        def handle_accept(self):

            conn, addr = self.accept()

            HTTPChannel(self, conn, addr)
```

```
        def handle_request(self, channel, method, path,
    header):
```

```
            try:

                # this is not safe!

                while path[:1] == "/":

                    path = path[1:]

                filename = os.path.join(ROOT, path)

                print path, "=>", filename

                file = open(filename, "r")

            except IOError:
```

```
            channel.pushstatus(404, "Not found")

            channel.push("Content-type:
text/html\r\n")

            channel.push("\r\n")

            channel.push("<html><body>File not
found.</body></html>\r\n")

        else:

            channel.pushstatus(200, "OK")

            channel.push("Content-type:
text/html\r\n")

            channel.push("\r\n")
```

```
channel.push_with_producer(FileProducer(file))



#

# try it out



s = HTTPServer(PORT)

print "serving at port", PORT
```

```
asyncore.loop()



serving at port 8000

log: adding channel <HTTPServer  at 8e54d0>

log: adding channel <HTTPChannel  at 8e64a0>

samples/sample.htm => .\samples/sample.htm

log: closing channel 96:<HTTPChannel connected at
8e64a0>
```

7.6. urllib 模块

urlib 模块为 HTTP，FTP，以及 gopher 提供了一个统一的客户端接口. 它会自动地根据 URL 选择合适的协议处理器.

从 URL 获取数据是非常简单的. 只需要调用 urlopen 方法, 然后从返回的流对象中读取数据即可, 如 Example 7-14 所示.

7.6.0.1. Example 7-14. 使用 urllib 模块获取远程资源

```
File: urllib-example-1.py



import urllib
```

```
fp = urllib.urlopen("http://www.python.org")



op = open("out.html", "wb")



n = 0



while 1:

    s = fp.read(8192)

    if not s:

        break

    op.write(s)

    n = n + len(s)



fp.close()

op.close()



for k, v in fp.headers.items():
```

```
    print k, "=", v



print "copied", n, "bytes from", fp.url



server = Apache/1.3.6 (Unix)

content-type = text/html

accept-ranges = bytes

date = Mon, 11 Oct 1999 20:11:40 GMT
```

```
connection = close
```

```
etag = "741e9-7870-37f356bf"

content-length = 30832

last-modified = Thu, 30 Sep 1999 12:25:35 GMT

copied 30832 bytes from http://www.python.org
```

这个流对象提供了一些非标准的属性. `headers` 是一个 *Message* 对象(在 `mimetools` 模块中定义), `url` 是实际的 URL . 后者会根据服务器的重定向而更新.

`urlopen` 函数实际上是一个辅助函数, 它会创建一个 *FancyURLopener* 类的实例并调用它的 `open` 方法. 你也可以继承这个类来完成特殊的行为. 例如 Example 7-15 中的类会自动地在必要时登陆服务器.

7.6.0.2. Example 7-15. 用 urllib 模块实现自动身份验证

```
File: urllib-example-3.py



import urllib



class myURLOpener(urllib.FancyURLopener):

    # read an URL, with automatic HTTP
authentication
```

```
    def setpasswd(self, user, passwd):

        self._ _user = user

        self._ _passwd = passwd



    def prompt_user_passwd(self, host, realm):

        return self._ _user, self._ _passwd



urlopener = myURLOpener()

urlopener.setpasswd("mulder", "trustno1")
```

```
fp = urlopener.open("http://www.secretlabs.com")

print fp.read()
```

7.7. urlparse 模块

urlparse 模块包含用于处理 URL 的函数, 可以在 URL 和平台特定的文件名间相互转换. 如 Example 7-16 所示.

7.7.0.1. Example 7-16. 使用 urlparse 模块

```
File: urlparse-example-1.py

import urlparse

print
urlparse.urlparse("http://host/path;params?query#
fragment")

('http', 'host', '/path', 'params', 'query',
'fragment')
```

一个常见用途就是把 HTTP URL 分割为主机名和路径组件(一个 HTTP 请求会涉及到主机名以及请求路径), 如 Example 7-17 所示.

7.7.0.2. Example 7-17. 使用 urlparse 模块处理 HTTP 定位器(HTTP Locators)

```
File: urlparse-example-2.py



import urlparse



scheme, host, path, params, query, fragment =\
```

```
urlparse.urlparse("http://host/path;params?query#
fragment")
```

```


if scheme == "http":

    print "host", "=>", host

    if params:

        path = path + ";" + params

    if query:

        path = path + "?" + query

    print "path", "=>", path
```

```
host => host

path => /path;params?query
```

Example 7-18 展示了如何使用 urlunparse 函数将各组成部分合并回一个 URL .

7.7.0.3. Example 7-18. 使用 urlparse 模块处理 HTTP 定位器(HTTP Locators)

```
File: urlparse-example-3.py
```

```

```

```
import urlparse



scheme, host, path, params, query, fragment =\

urlparse.urlparse("http://host/path;params?query#
fragment")



if scheme == "http":

    print "host", "=>", host

    print "path", "=>", urlparse.urlunparse(

    (None, None, path, params, query, None)
```

```
    )



host => host

path => /path;params?query
```

Example 7-19 使用 urljoin 函数将绝对路径和相对路径组合起来.

7.7.0.4. Example 7-19. 使用 urlparse 模块组合相对定位器

```
File: urlparse-example-4.py



import urlparse



base = "http://spam.egg/my/little/pony"



for path in "/index", "goldfish", "../black/cat":

    print path, "=>", urlparse.urljoin(base, path)



/index => http://spam.egg/index

goldfish => http://spam.egg/my/little/goldfish
```

```
../black/cat => http://spam.egg/my/black/cat
```

7.8. cookie 模块

(2.0 中新增) 该模块为 HTTP 客户端和服务器提供了基本的 cookie 支持. Example 7-20 展示了它的使用.

7.8.0.1. Example 7-20. 使用 cookie 模块

```
File: cookie-example-1.py
```

```

```

```
import Cookie

import os, time



cookie = Cookie.SimpleCookie()

cookie["user"] = "Mimi"

cookie["timestamp"] = time.time()



print cookie
```

```
# simulate CGI roundtrip

os.environ["HTTP_COOKIE"] = str(cookie)



print



cookie = Cookie.SmartCookie()

cookie.load(os.environ["HTTP_COOKIE"])
```

```

```

```
for key, item in cookie.items():

    # dictionary items are "Morsel" instances

    # use value attribute to get actual value

    print key, repr(item.value)



Set-Cookie: timestamp=736513200;

Set-Cookie: user=Mimi;
```

```
user 'Mimi'

timestamp '736513200'
```

7.9. robotparser 模块

(2.0 中新增) robotparser 模块用来读取 robots.txt 文件, 该文件用于 Robot Exclusion Protocol (搜索机器人排除协议? http://info.webcrawler.com/mak/projects/robots/robots.html).

如果你实现的一个 HTTP 机器人会访问网路上的任意站点(并不只是你自己的站点), 那么最好还是用该模块检查下你所做的一切是不是受欢迎的. Example 7-21 展示了该模块的使用.

7.9.0.1. Example 7-21. 使用 robotparser 模块

```
File: robotparser-example-1.py



import robotparser



r = robotparser.RobotFileParser()

r.set_url("http://www.python.org/robots.txt")

r.read()



if r.can_fetch("*", "/index.html"):
```

```
    print "may fetch the home page"



if r.can_fetch("*", "/tim_one/index.html"):

    print "may fetch the tim peters archive"



may fetch the home page
```

7.10. ftplib 模块

ftplib 模块包含了一个 File Transfer Protocol (FTP , 文件传输协议)客户端的实现.

Example 7-22 展示了如何登陆并获得登陆目录的文件列表. 注意这里的文件列表 (列目录操作)格式与服务器有关(一般和主机平台的列目录工具输出格式相同, 例如 Unix 下的 ls 和 Windows/DOS 下的 dir).

7.10.0.1. Example 7-22. 使用 ftplib 模块获得目录列表

```
File: ftplib-example-1.py



import ftplib



ftp = ftplib.FTP("www.python.org")
```

```
ftp.login("anonymous", "ftplib-example-1")



print ftp.dir()



ftp.quit()



total 34
```

```
drwxrwxr-x  11 root     4127         512 Sep 14
14:18 .
```

```
drwxrwxr-x  11 root     4127         512 Sep 14
14:18 ..

drwxrwxr-x   2 root     4127         512 Sep 13
15:18 RCS

lrwxrwxrwx   1 root     bin           11 Jun 29
14:34 README -> welcome.msg

drwxr-xr-x   3 root     wheel        512 May 19
1998 bin

drwxr-sr-x   3 root     1400         512 Jun  9
1997 dev

drwxrwxr--   2 root     4127         512 Feb  8
1998 dup
```

```
drwxr-xr-x   3 root     wheel        512 May 19
1998 etc


...
```

下载文件很简单；使用合适的 `retr` 函数即可. 注意当你下载文本文件时，你必须自己加上行结束符. Example 7-23 中使用了一个 `lambda` 表达式完成这项工作.

7.10.0.2. Example 7-23. 使用 ftplib 模块下载文件

```
File: ftplib-example-2.py
```

```

```

```
import ftplib

import sys



def gettext(ftp, filename, outfile=None):

    # fetch a text file

    if outfile is None:

        outfile = sys.stdout

    # use a lambda to add newlines to the lines read
from the server
```

```
    ftp.retrlines("RETR " + filename, lambda s,
w=outfile.write: w(s+"\n"))



def getbinary(ftp, filename, outfile=None):

    # fetch a binary file

    if outfile is None:

        outfile = sys.stdout

    ftp.retrbinary("RETR " + filename,
outfile.write)
```

```
ftp = ftplib.FTP("www.python.org")

ftp.login("anonymous", "ftplib-example-2")



gettext(ftp, "README")

getbinary(ftp, "welcome.msg")



WELCOME to python.org, the Python programming
language home site.
```

```
You are number %N of %M allowed users.  Ni!



Python Web site: http://www.python.org/



CONFUSED FTP CLIENT?  Try begining your login
password with '-' dash.

This turns off continuation messages that may be
confusing your client.

...
```

最后, Example 7-24 将文件复制到 FTP 服务器上. 这个脚本使用文件扩展名来判断文件是文本文件还是二进制文件.

7.10.0.3. Example 7-24. 使用 ftplib 模块上传文件

```
File: ftplib-example-3.py



import ftplib

import os



def upload(ftp, file):

    ext = os.path.splitext(file)[1]
```

```
    if ext in (".txt", ".htm", ".html"):
        ftp.storlines("STOR " + file, open(file))
    else:
        ftp.storbinary("STOR " + file, open(file,
"rb"), 1024)



ftp = ftplib.FTP("ftp.fbi.gov")
ftp.login("mulder", "trustno1")
```

```
upload(ftp, "trixie.zip")
```

```
upload(ftp, "file.txt")
upload(ftp, "sightings.jpg")
```

7.11. gopherlib 模块

gopherlib 模块包含了一个 gopher 客户端实现, 如 Example 7-25 所示.

7.11.0.1. Example 7-25. 使用 gopherlib 模块

```
File: gopherlib-example-1.py
```

```
import gopherlib



host = "gopher.spam.egg"



f = gopherlib.send_selector("1/", host)

for item in gopherlib.get_directory(f):

    print item
```

```
['0', "About Spam.Egg's Gopher Server", "0/About's
Spam.Egg's
```

```
Gopher Server", 'gopher.spam.egg', '70', '+']


['1', 'About Spam.Egg', '1/Spam.Egg',
'gopher.spam.egg', '70', '+']


['1', 'Misc', '1/Misc', 'gopher.spam.egg', '70',
'+']


...
```

7.12. httplib 模块

httplib 模块提供了一个 HTTP 客户端接口, 如 Example 7-26 所示.

7.12.0.1. Example 7-26. 使用 httplib 模块

```
File: httplib-example-1.py



import httplib



USER_AGENT = "httplib-example-1.py"



class Error:
```

```
    # indicates an HTTP error
```

```
    def _ _init_ _(self, url, errcode, errmsg,
headers):

        self.url = url

        self.errcode = errcode

        self.errmsg = errmsg

        self.headers = headers

    def _ _repr_ _(self):
```

```
        return (
            "<Error for %s: %s %s>" %
            (self.url, self.errcode, self.errmsg)
            )


class Server:


    def _ _init_ _(self, host):
        self.host = host
```

```
    def fetch(self, path):
```

```
        http = httplib.HTTP(self.host)


        # write header
        http.putrequest("GET", path)
        http.putheader("User-Agent", USER_AGENT)
        http.putheader("Host", self.host)
```

```
            http.putheader("Accept", "*/*")
            http.endheaders()

            # get response
            errcode, errmsg, headers = http.getreply()

            if errcode != 200:
                raise Error(errcode, errmsg, headers)

            file = http.getfile()
            return file.read()

    if __name__ == "__main__":

        server = Server("www.pythonware.com")
        print server.fetch("/index.htm")
```

注意 httplib 提供的 HTTP 客户端在等待服务器回复的时候会阻塞程序. 异步的解决方法请参阅 asyncore 模块中的例子.

7.12.1. 将数据发送给服务器

httplib 可以用来发送其他 HTTP 命令, 例如 POST , 如 Example 7-27 所示.

7.12.1.1. Example 7-27. 使用 httplib 发送数据

```
File: httplib-example-2.py



import httplib



USER_AGENT = "httplib-example-2.py"
```

```
def post(host, path, data, type=None):
```

```
    http = httplib.HTTP(host)



    # write header

    http.putrequest("PUT", path)
```

```
    http.putheader("User-Agent", USER_AGENT)

    http.putheader("Host", host)

    if type:

        http.putheader("Content-Type", type)

    http.putheader("Content-Length",
str(len(size)))

    http.endheaders()



    # write body

    http.send(data)



    # get response
```

```
    errcode, errmsg, headers = http.getreply()
```

```


    if errcode != 200:

        raise Error(errcode, errmsg, headers)


```

```
    file = http.getfile()

    return file.read()



if _ _name_ _ == "_ _main_ _":



    post("www.spam.egg", "/bacon.htm", "a piece of
data", "text/plain")
```

7.13. poplib 模块

poplib 模块(如 Example 7-28 所示) 提供了一个 Post Office Protocol (POP3 协议) 客户端实现. 这个协议用来从邮件服务器 "pop" (拷贝) 信息到你的个人电脑.

7.13.0.1. Example 7-28. 使用 poplib 模块

```
File: poplib-example-1.py
```

```

```

```
import poplib

import string, random

import StringIO, rfc822
```

```
SERVER = "pop.spam.egg"



USER  = "mulder"

PASSWORD = "trustno1"



# connect to server

server = poplib.POP3(SERVER)



# login

server.user(USER)

server.pass_(PASSWORD)
```

```
# list items on server
```

```
resp, items, octets = server.list()
```

```
# download a random message
```

```
id, size = string.split(random.choice(items))

resp, text, octets = server.retr(id)



text = string.join(text, "\n")

file = StringIO.StringIO(text)



message = rfc822.Message(file)



for k, v in message.items():

    print k, "=", v



print message.fp.read()
```

```
subject = ANN: (the eff-bot guide to) The Standard
Python Library
```

```
message-id = <199910120808.KAA09206@spam.egg>

received = (from fredrik@spam.egg)
```

```
 by spam.egg (8.8.7/8.8.5) id KAA09206

 for mulder; Tue, 12 Oct 1999 10:08:47 +0200

from = Fredrik Lundh <fredrik@spam.egg>

date = Tue, 12 Oct 1999 10:08:47 +0200

to = mulder@spam.egg



...
```

7.14. imaplib 模块

imaplib 模块提供了一个 Internet Message Access Protocol (IMAP, Internet 消息访问协议) 的客户端实现. 这个协议允许你访问邮件服务器的邮件目录, 就好像是在本机访问一样. 如 Example 7-29 所示.

7.14.0.1. Example 7-29. 使用 imaplib 模块

```
File: imaplib-example-1.py



```

```
import imaplib
```

```
import string, random

import StringIO, rfc822
```

```
SERVER = "imap.spam.egg"



USER  = "mulder"

PASSWORD = "trustno1"



# connect to server

server = imaplib.IMAP4(SERVER)



# login

server.login(USER, PASSWORD)

server.select()



# list items on server
```

```
resp, items = server.search(None, "ALL")
```

```
items = string.split(items[0])
```

```
# fetch a random item

id = random.choice(items)

resp, data = server.fetch(id, "(RFC822)")

text = data[0][1]



file = StringIO.StringIO(text)



message = rfc822.Message(file)



for k, v in message.items():

    print k, "=", v



print message.fp.read()
```

```
server.logout()
```

```
subject = ANN: (the eff-bot guide to) The Standard
Python Library


message-id =
<199910120816.KAA12177@larch.spam.egg>


to = mulder@spam.egg


date = Tue, 12 Oct 1999 10:16:19 +0200 (MET DST)


from = <effbot@spam.egg>


received = (effbot@spam.egg) by imap.algonet.se
(8.8.8+Sun/8.6.12)


id KAA12177 for effbot@spam.egg; Tue, 12 Oct 1999
10:16:19 +0200


(MET DST)




body text for test 5
```

7.15. smtplib 模块

smtplib 模块提供了一个 Simple Mail Transfer Protocol（SMTP，简单邮件传输协议）客户端实现. 该协议用于通过 Unix 邮件服务器发送邮件, 如 Example 7-30 所示.

读取邮件请使用 poplib 或 imaplib 模块.

7.15.0.1. Example 7-30. 使用 smtplib 模块

```
File: smtplib-example-1.py



import smtplib

import string, sys



HOST = "localhost"



FROM = "effbot@spam.egg"

TO = "fredrik@spam.egg"



SUBJECT = "for your information!"



BODY = "next week: how to fling an otter"



```

```
body = string.join((

```

```
    "From: %s" % FROM,

    "To: %s" % TO,
```

```
    "Subject: %s" % SUBJECT,
    "",
    BODY), "\r\n")

print body

server = smtplib.SMTP(HOST)
server.sendmail(FROM, [TO], body)
server.quit()
```

```
From: effbot@spam.egg
To: fredrik@spam.egg
Subject: for your information!

next week: how to fling an otter
```

7.16. telnetlib 模块

`telnetlib` 模块提供了一个 telnet 客户端实现.

Example 7-31 连接到一台 Unix 计算机, 登陆, 然后请求一个目录的列表.

7.16.0.1. Example 7-31. 使用 telnetlib 模块登陆到远程服务器

```
File: telnetlib-example-1.py



import telnetlib

import sys



HOST = "spam.egg"



USER = "mulder"

PASSWORD = "trustno1"



telnet = telnetlib.Telnet(HOST)
```

```

```

```
telnet.read_until("login: ")
```

```
telnet.write(USER + "\n")



telnet.read_until("Password: ")

telnet.write(PASSWORD + "\n")



telnet.write("ls librarybook\n")

telnet.write("exit\n")



print telnet.read_all()



[spam.egg mulder]$ ls

README
os-path-isabs-example-1.py

SimpleAsyncHTTP.py
os-path-isdir-example-1.py

aifc-example-1.py
os-path-isfile-example-1.py

anydbm-example-1.py
os-path-islink-example-1.py
```

```
array-example-1.py
os-path-ismount-example-1.py

...
```

7.17. nntplib 模块

nntplib 模块提供了一个网络新闻传输协议(Network News Transfer Protocol, NNTP)客户端的实现.

7.17.1. 列出消息

从新闻服务器上读取消息之前, 你必须连接这个服务器并选择一个新闻组. Example 7-32 中的脚本会从服务器下载一个完成的消息列表, 然后根据列表做简单的统计.

7.17.1.1. Example 7-32. 使用 nntplib 模块列出消息

```
File: nntplib-example-1.py



import nntplib

import string



SERVER = "news.spam.egg"
```

```
GROUP  = "comp.lang.python"
```

```
AUTHOR = "fredrik@pythonware.com" # eff-bots human
alias



# connect to server

server = nntplib.NNTP(SERVER)



# choose a newsgroup

resp, count, first, last, name = server.group(GROUP)

print "count", "=>", count

print "range", "=>", first, last



# list all items on the server

resp, items = server.xover(first, last)



# extract some statistics

authors = {}

subjects = {}
```

```
for id, subject, author, date, message_id,
references, size, lines in items:

    authors[author] = None

    if subject[:4] == "Re: ":

        subject = subject[4:]

    subjects[subject] = None

    if string.find(author, AUTHOR) >= 0:

        print id, subject



print "authors", "=>", len(authors)

print "subjects", "=>", len(subjects)
```

```
count => 607

range => 57179 57971

57474 Three decades of Python!

...

57477 More Python books coming...

authors => 257
```

```
subjects => 200
```

7.17.2. 下载消息

下载消息是很简单的，只需要调用 `article` 方法，如 Example 7-33 所示.

7.17.2.1. Example 7-33. 使用 nntplib 模块下载消息

```
File: nntplib-example-2.py



import nntplib

import string



SERVER = "news.spam.egg"

GROUP  = "comp.lang.python"

KEYWORD = "tkinter"



# connect to server

server = nntplib.NNTP(SERVER)



resp, count, first, last, name = server.group(GROUP)
```

```
resp, items = server.xover(first, last)

for id, subject, author, date, message_id,
references, size, lines in items:

    if string.find(string.lower(subject),
KEYWORD) >= 0:

        resp, id, message_id, text =
server.article(id)

        print author

        print subject

        print len(text), "lines in article"



"Fredrik Lundh" <fredrik@pythonware.com>

Re: Programming Tkinter (In Python)

110 lines in article

...
```

Example 7-34 展示了如何进一步处理这些消息, 你可以把它封装到一个 *Message* 对象中(使用 rfc822 模块).

7.17.2.2. Example 7-34. 使用 nntplib 和 rfc822 模块处理消息

```
File: nntplib-example-3.py
```

```
import nntplib

import string, random

import StringIO, rfc822



SERVER = "news.spam.egg"

GROUP  = "comp.lang.python"



# connect to server

server = nntplib.NNTP(SERVER)



resp, count, first, last, name = server.group(GROUP)

for i in range(10):

    try:

        id = random.randint(int(first), int(last))

        resp, id, message_id, text =
server.article(str(id))
```

```
    except (nntplib.error_temp,
nntplib.error_perm):
```

```
        pass # no such message (maybe it was
deleted?)

    else:

        break # found a message!

else:

    raise SystemExit



text = string.join(text, "\n")

file = StringIO.StringIO(text)



message = rfc822.Message(file)



for k, v in message.items():

    print k, "=", v



print message.fp.read()
```

```
mime-version = 1.0

content-type = text/plain; charset="iso-8859-1"

message-id =
<008501bf1417$1cf90b70$f29b12c2@sausage.spam.egg>

lines = 22

...

from = "Fredrik Lundh" <fredrik@pythonware.com>

nntp-posting-host = parrot.python.org

subject = ANN: (the eff-bot guide to) The Standard
Python Library

...

</F>
```

到这一步后，你可以使用 `htmllib` ， `uu` ， 以及 `base64` 继续处理这些消息.

7.18. SocketServer 模块

`SocketServer` 为各种基于 socket 的服务器提供了一个框架. 该模块提供了大量的类，你可以用它们来创建不同的服务器.

Example 7-35 使用该模块实现了一个 Internet 时间协议服务器. 你可以用前边的 timeclient 脚本连接它.

7.18.0.1. Example 7-35. 使用 SocketServer 模块

```
File: socketserver-example-1.py



import SocketServer

import time



# user-accessible port

PORT = 8037



# reference time

TIME1970 = 2208988800L



class
TimeRequestHandler(SocketServer.StreamRequestHand
ler):

    def handle(self):

        print "connection from",
self.client_address

        t = int(time.time()) + TIME1970
```

```
        b = chr(t>>24&255) + chr(t>>16&255) +
chr(t>>8&255) + chr(t&255)

        self.wfile.write(b)



server = SocketServer.TCPServer(("", PORT),
TimeRequestHandler)

print "listening on port", PORT

server.serve_forever()



connection from ('127.0.0.1', 1488)

connection from ('127.0.0.1', 1489)

...
```

7.19. BaseHTTPServer 模块

这是一个建立在 SocketServer 框架上的基本框架, 用于 HTTP 服务器.

Example 7-36 在每次重新载入页面时会生成一条随机信息. path 变量包含当前 URL , 你可以使用它为不同的 URL 生成不同的内容 (访问除根目录的其他任何 path 该脚本都会返回一个错误页面).

7.19.0.1. Example 7-36. 使用 BaseHTTPServer 模块

```
File: basehttpserver-example-1.py
```

```
import BaseHTTPServer

import cgi, random, sys



MESSAGES = [

    "That's as maybe, it's still a frog.",

    "Albatross! Albatross! Albatross!",

    "It's Wolfgang Amadeus Mozart.",

    "A pink form from Reading.",

    "Hello people, and welcome to 'It's a Tree.'"

    "I simply stare at the brick and it goes to
sleep.",

]



class
Handler(BaseHTTPServer.BaseHTTPRequestHandler):



    def do_GET(self):

        if self.path != "/":
```

```
            self.send_error(404, "File not found")
            return
        self.send_response(200)
        self.send_header("Content-type",
"text/html")
        self.end_headers()
        try:
            # redirect stdout to client
            stdout = sys.stdout
            sys.stdout = self.wfile
            self.makepage()
        finally:
            sys.stdout = stdout # restore


    def makepage(self):
        # generate a random message
        tagline = random.choice(MESSAGES)
        print "<html>"
```

```
            print "<body>"

            print "<p>Today's quote: "

            print "<i>%s</i>" % cgi.escape(tagline)

            print "</body>"

            print "</html>"



    PORT = 8000



    httpd = BaseHTTPServer.HTTPServer(("", PORT),
    Handler)

    print "serving at port", PORT

    httpd.serve_forever()
```

更有扩展性的 HTTP 框架请参阅 SimpleHTTPServer 和 CGIHTTPServer 模块.

7.20. SimpleHTTPServer 模块

SimpleHTTPServer 模块是一个简单的 HTTP 服务器, 它提供了标准的 GET 和 HEAD 请求处理器. 客户端请求的路径名称会被翻译为一个相对文件名(相对于服务器启动时的当前路径). Example 7-37 展示了该模块的使用.

7.20.0.1. Example 7-37. 使用 SimpleHTTPServer 模块

File: simplehttpserver-example-1.py

```
import SimpleHTTPServer

import SocketServer



# minimal web server.  serves files relative to the

# current directory.



PORT = 8000



Handler =
SimpleHTTPServer.SimpleHTTPRequestHandler



httpd = SocketServer.TCPServer(("", PORT), Handler)



print "serving at port", PORT

httpd.serve_forever()
```

```
serving at port 8000

localhost - - [11/Oct/1999 15:07:44] code 403,
message Directory listing not sup

ported

localhost - - [11/Oct/1999 15:07:44] "GET /
HTTP/1.1" 403 -

localhost - - [11/Oct/1999 15:07:56] "GET
/samples/sample.htm HTTP/1.1" 200 -
```

这个服务器会忽略驱动器符号和相对路径名(例如 `..`). 但它并没有任何访问验证处理, 所以请小心使用.

Example 7-38 实现了个迷你的 web 代理. 发送给代理的 HTTP 请求必须包含目标服务器的完整 URI . 代理服务器使用 `urllib` 来获取目标服务器的数据.

7.20.0.2. Example 7-38. 使用 SimpleHTTPServer 模块实现代理

```
File: simplehttpserver-example-2.py



# a truly minimal HTTP proxy



import SocketServer

import SimpleHTTPServer
```

```
import urllib



PORT = 1234



class
Proxy(SimpleHTTPServer.SimpleHTTPRequestHandler):

    def do_GET(self):

        self.copyfile(urllib.urlopen(self.path),
self.wfile)



httpd = SocketServer.ForkingTCPServer(('', PORT),
Proxy)

print "serving at port", PORT

httpd.serve_forever()
```

7.21. CGIHTTPServer 模块

CGIHTTPServer 模块是一个可以通过公共网关接口(common gateway interface , CGI)调用外部脚本的 HTTP 服务器. 如 Example 7-39 所示.

7.21.0.1. Example 7-39. 使用 CGIHTTPServer 模块

```
File: cgihttpserver-example-1.py
```

```
import CGIHTTPServer

import BaseHTTPServer



class
Handler(CGIHTTPServer.CGIHTTPRequestHandler):

    cgi_directories = ["/cgi"]



PORT = 8000



httpd = BaseHTTPServer.HTTPServer(("", PORT),
Handler)

print "serving at port", PORT

httpd.serve_forever()
```

7.22. cgi 模块

cgi 模块为 CGI 脚本提供了函数和类支持. 它还可以处理 CGI 表单数据.

Example 7-40 展示了一个简单的 CGI 脚本, 它返回给定目录下的文件列表(相对于脚本中指定的根目录)

7.22.0.1. Example 7-40. 使用 cgi 模块

```
File: cgi-example-1.py



import cgi

import os, urllib



ROOT = "samples"



# header

print "text/html"

print



query = os.environ.get("QUERY_STRING")

if not query:

    query = "."



script = os.environ.get("SCRIPT_NAME", "")

if not script:
```

```
    script = "cgi-example-1.py"



print "<html>"

print "<head>"

print "<title>file listing</title>"

print "</head>"

print "</html>"



print "<body>"



try:

    files = os.listdir(os.path.join(ROOT, query))

except os.error:

    files = []



for file in files:

    link = cgi.escape(file)
```

```
    if os.path.isdir(os.path.join(ROOT, query,
file)):

        href = script + "?" + os.path.join(query,
file)

        print "<p><a href='%s'>%s</a>" % (href,
cgi.escape(link))

    else:

        print "<p>%s" % link



print "</body>"

print "</html>"
```

```
text/html
```

```
<html>

<head>

<title>file listing</title>
```

```
</head>
```

```
</html>
```

```
<body>

<p>sample.gif

<p>sample.gz

<p>sample.netrc

...

<p>sample.txt

<p>sample.xml

<p>sample~

<p><a href='cgi-example-1.py?web'>web</a>

</body>

</html>
```

7.23. webbrowser 模块

(2.0 中新增) webbrowser 模块提供了一个到系统标准 web 浏览器的接口. 它提供了一个 open 函数, 接受文件名或 URL 作为参数, 然后在浏览器中打开它. 如果你又一次调用 open 函数, 那么它会尝试在相同的窗口打开新页面. 如 Example 7-41 所示.

7.23.0.1. Example 7-41. 使用 webbrowser 模块

```
File: webbrowser-example-1.py
```

```
import webbrowser

import time



webbrowser.open("http://www.pythonware.com")



# wait a while, and then go to another page

time.sleep(5)

webbrowser.open(

"http://www.pythonware.com/people/fredrik/librarybook.htm"

    )
```

在 Unix 下, 该模块支持 lynx , Netscape , Mosaic , Konquerer , 和 Grail . 在 Windows 和 Macintosh 下, 它会调用标准浏览器 (在注册表或是 Internet 选项面板中定义).

8. 国际化

8.1. locale 模块

locale 模块提供了 C 本地化(localization)函数的接口, 如 Example 8-1 所示. 同时提供相关函数, 实现基于当前 locale 设置的数字, 字符串转换. (而 int , float , 以及 string 模块中的相关转换函数不受 locale 设置的影响.)

====Example 8-1. 使用 locale 模块格式化数据=====[eg-8-1]

```
File: locale-example-1.py



import locale



print "locale", "=>",
locale.setlocale(locale.LC_ALL, "")



# integer formatting

value = 4711

print locale.format("%d", value, 1), "==",

print locale.atoi(locale.format("%d", value, 1))
```

```

```

```
# floating point

value = 47.11

print locale.format("%f", value, 1), "==",
```

```
print locale.atof(locale.format("%f", value, 1))



info = locale.localeconv()

print info["int_curr_symbol"]



locale => Swedish_Sweden.1252

4,711 == 4711

47,110000 == 47.11

SEK
```

Example 8-2 展示了如何使用 `locale` 模块获得当前平台 locale 设置.

8.1.0.1. Example 8-2. 使用 locale 模块获得当前平台 locale 设置

```
File: locale-example-2.py
```

```
import locale
```

```


language, encoding = locale.getdefaultlocale()
```

```
print "language", language

print "encoding", encoding
```

```
language sv_SE

encoding cp1252
```

8.2. unicodedata 模块

(2.0 中新增) unicodedata 模块包含了 Unicode 字符的属性, 例如字符类别, 分解数据, 以及数值. 如 Example 8-3 所示.

8.2.0.1. Example 8-3. 使用 unicodedata 模块

```
File: unicodedata-example-1.py

import unicodedata
```

```

for char in [u"A", u"-", u"1", u"\N{LATIN CAPITAL
LETTER O WITH DIAERESIS}"]:

    print repr(char),
```

```
    print unicodedata.category(char),

    print repr(unicodedata.decomposition(char)),



    print unicodedata.decimal(char, None),

    print unicodedata.numeric(char, None)



u'A' Lu '' None None

u'-' Pd '' None None

u'1' Nd '' 1 1.0

u'\303\226' Lu '004F 0308' None None
```

在 Python 2.0 中缺少 CJK 象形文字和韩语音节的属性. 这影响到了 0x3400-0x4DB5 , 0x4E00-0x9FA5 , 以及 0xAC00-D7A3 中的字符, 不过每个区间内的第一个字符属性是正确的, 我们可以把字符映射到起始实现正常操作:

```
def remap(char):

    # fix for broken unicode property database in
Python 2.0

    c = ord(char)

    if 0x3400 <= c <= 0x4DB5:
```

```
        return unichr(0x3400)

    if 0x4E00 <= c <= 0x9FA5:

        return unichr(0x4E00)

    if 0xAC00 <= c <= 0xD7A3:

        return unichr(0xAC00)

    return char
```

Python 2.1 修复了这个 bug .

8.3. ucnhash 模块

(仅适用于 2.0)ucnhash 模块为一些 Unicode 字符代码提供了特定的命名. 你可以直接使用 \N{} 转义符将 Unicode 字符名称映射到字符代码上. 如 Example 8-4 所示.

8.3.0.1. Example 8-4. 使用 ucnhash 模块

```
File: ucnhash-example-1.py
```

```
# Python imports this module automatically, when it
sees
```

```
# the first \N{} escape

# import ucnhash
```

```
print repr(u"\N{FROWN}")

print repr(u"\N{SMILE}")

print repr(u"\N{SKULL AND CROSSBONES}")
```

```
u'\u2322'

u'\u2323'

u'\u2620'
```

9. 多媒体相关模块

"Wot? No quote?"
- Guido van Rossum

9.1. 概览

Python 提供了一些用于处理图片和音频文件的模块.

另请参阅 *Pythonware Image Library* (PIL , http://www.pythonware.com/products/pil/), 以及 *PythonWare Sound Toolkit* (PST , http://www.pythonware.com/products/pst/).

译注: 别参阅 PST 了, 废了, 用 pymedia 代替吧.

9.2. imghdr 模块

imghdr 模块可识别不同格式的图片文件. 当前版本可以识别 bmp，gif，jpeg，pbm，pgm，png，ppm，rast (Sun raster)，rgb (SGI)，tiff，以及 xbm 图像. 如 Example 9-1 所示.

9.2.0.1. Example 9-1. 使用 imghdr 模块

```
File: imghdr-example-1.py



import imghdr



result = imghdr.what("samples/sample.jpg")



if result:

    print "file format:", result

else:

    print "cannot identify file"
```

```

```

```
file format: jpeg

# 使用 PIL

import Image
```

```
im = Image.open("samples/sample.jpg")

print im.format, im.mode, im.size
```

9.3. sndhdr 模块

sndhdr 模块，可来识别不同的音频文件格式，并提取文件内容相关信息. 如 Example 9-2 所示.

执行成功后，what 函数将返回一个由文件类型，采样频率，声道数，音轨数和每个采样点位数组成的元组. 具体含义请参考 help(sndhdr) .

9.3.0.1. Example 9-2. 使用 sndhdr 模块

```
File: sndhdr-example-1.py

import sndhdr
```

```
result = sndhdr.what("samples/sample.wav")
```

```
if result:

    print "file format:", result
```

```
else:

    print "cannot identify file"



file format: ('wav', 44100, 1, -1, 16)
```

9.4. whatsound 模块

(已废弃) whatsound 是 sndhdr 模块的一个别名. 如 Example 9-3 所示.

9.4.0.1. Example 9-3. 使用 whatsound 模块

```
File: whatsound-example-1.py



import whatsound # same as sndhdr



result = whatsound.what("samples/sample.wav")



if result:

    print "file format:", result
```

```
else:

    print "cannot identify file"



file format: ('wav', 44100, 1, -1, 16)
```

9.5. aifc 模块

aifc 模块用于读写 AIFF 和 AIFC 音频文件(在 SGI 和 Macintosh 的计算机上使用). 如 Example 9-4 所示.

9.5.0.1. Example 9-4. 使用 aifc 模块

```
File: SimpleAsyncHTTP.py



import asyncore

import string, socket
```

```
import StringIO
```

```
import mimetools, urlparse



class AsyncHTTP(asyncore.dispatcher_with_send):
```

```
    # HTTP requestor



    def _ _init_ _(self, uri, consumer):

        asyncore.dispatcher_with_send._ _init_
_(self)



        self.uri = uri

        self.consumer = consumer



        # turn the uri into a valid request

        scheme, host, path, params, query, fragment
= urlparse.urlparse(uri)

        assert scheme == "http", "only supports HTTP
requests"

        try:
```

```
            host, port = string.split(host, ":", 1)
```

```
            port = int(port)

        except (TypeError, ValueError):

            port = 80 # default port
```

```
            if not path:
                path = "/"
            if params:
                path = path + ";" + params
            if query:
                path = path + "?" + query

            self.request = "GET %s
    HTTP/1.0\r\nHost: %s\r\n\r\n" % (path, host)

            self.host = host
            self.port = port

            self.status = None
            self.header = None

            self.data = ""
```

```
        # get things going!

        self.create_socket(socket.AF_INET,
socket.SOCK_STREAM)

        self.connect((host, port))



    def handle_connect(self):

        # connection succeeded

        self.send(self.request)



    def handle_expt(self):

        # connection failed; notify consumer
(status is None)

        self.close()

        try:
```

```
            http_header =
self.consumer.http_header
```

```
        except AttributeError:

            pass
```

```
            else:
                http_header(self)


        def handle_read(self):
            data = self.recv(2048)
            if not self.header:
                self.data = self.data + data
                try:
                    i = string.index(self.data,
"\r\n\r\n")
                except ValueError:
                    return # continue
                else:
                    # parse header
                    fp =
StringIO.StringIO(self.data[:i+4])
                    # status line is "HTTP/version
status message"
                    status = fp.readline()
```

```
                self.status = string.split(status,
" ", 2)

                # followed by a rfc822-style
message header

                self.header =
mimetools.Message(fp)

                # followed by a newline, and the
payload (if any)

                data = self.data[i+4:]

                self.data = ""

                # notify consumer (status is
non-zero)

                try:

                    http_header =
self.consumer.http_header

                except AttributeError:

                    pass
```

```
                else:
```

```
                    http_header(self)

                if not self.connected:
```

```
                    return # channel was closed by
consumer



            self.consumer.feed(data)



        def handle_close(self):

            self.consumer.close()

            self.close()
```

9.6. sunau 模块

sunau 模块用于读写 Sun AU 音频文件. 如 Example 9-5 所示.

9.6.0.1. Example 9-5. 使用 sunau 模块

```
File: sunau-example-1.py



import sunau
```

```

```

```
w = sunau.open("samples/sample.au", "r")
```

```
if w.getnchannels() == 1:
    print "mono,",
else:
    print "stereo,",

print w.getsampwidth()*8, "bits,",
print w.getframerate(), "Hz sampling rate"
```

```
mono, 16 bits, 8012 Hz sampling rate
```

9.7. sunaudio 模块

sunaudio 模块用于识别 Sun AU 音频文件, 并提取其基本信息. sunau 模块为 Sun AU 文件提供了更完成的支持. 如 Example 9-6 所示

9.7.0.1. Example 9-6. 使用 sunaudio 模块

```
File: sunaudio-example-1.py

import sunaudio
```

```
file = "samples/sample.au"

print sunaudio.gethdr(open(file, "rb"))
```

```
(6761, 1, 8012, 1, 'sample.au')
```

9.8. wave 模块

wave 模块用于读写 Microsoft WAV 音频文件, 如 Example 9-7 所示.

9.8.0.1. Example 9-7. 使用 wave 模块

```
File: wave-example-1.py

import wave
```

```
w = wave.open("samples/sample.wav", "r")
```

```
if w.getnchannels() == 1:
```

```
    print "mono,",
else:
    print "stereo,",

print w.getsampwidth()*8, "bits,",
print w.getframerate(), "Hz sampling rate"

mono, 16 bits, 44100 Hz sampling rate
```

9.9. audiodev 模块

(只用于 Unix) audiodev 为 Sun 和 SGI 计算机提供了音频播放支持. 如 Example 9-8 所示.

9.9.0.1. Example 9-8. 使用 audiodev 模块

```
File: audiodev-example-1.py


import audiodev
import aifc
```

```
sound = aifc.open("samples/sample.aiff", "r")



player = audiodev.AudioDev()



player.setoutrate(sound.getframerate())

player.setsampwidth(sound.getsampwidth())

player.setnchannels(sound.getnchannels())



bytes_per_frame = sound.getsampwidth() *
sound.getnchannels()

bytes_per_second = sound.getframerate() *
bytes_per_frame



while 1:

    data = sound.readframes(bytes_per_second)

    if not data:

        break
```

```
        player.writeframes(data)



    player.wait()
```

9.10. winsound 模块

(只用于 Windows) winsound 模块允许你在 Winodws 平台上播放 Wave 文件. 如 Example 9-9 所示.

9.10.0.1. Example 9-9. 使用 winsound 模块

```
    File: winsound-example-1.py



    import winsound



    file = "samples/sample.wav"



    winsound.PlaySound(
```

```
        file,
```

```
        winsound.SND_FILENAME|winsound.SND_NOWAIT,
```

```
        )
```

flag 变量说明:

- SND_FILENAME - sound 是一个 wav 文件名
- SND_ALIAS - sound 是一个注册表中指定的别名
- SND_LOOP - 重复播放直到下一次 PlaySound ; 必须指定 SND_ASYNC
- SND_MEMORY - sound 是一个 wav 文件的内存映像
- SND_PURGE - 停止指定 sound 的所有实例
- SND_ASYNC - 异步播放声音, 声音开始播放后函数立即返回
- SND_NODEFAULT - 找不到 sound 时不播放默认的 beep 声音
- SND_NOSTOP - 不打断当前播放中的任何 sound
- SND_NOWAIT - sound 驱动忙时立即返回

---

10. 数据储存

"Unlike mainstream component programming, scripts usually do not introduce new components but simply 'wire' existing ones. Scripts can be seen as introducing behavior but no new state ... Of course, there is nothing to stop a 'scripting' language from introducing persistent state — it then simply turns into a normal programming language."
- Clemens Szyperski, in Component Software

10.1. 概览

Python 提供了多种相似数据库管理(database manager)的驱动, 它们的模型都基于 Unix 的 dbm 库. 这些数据库和普通的字典对象类似, 但这里需要注意的是它只能接受字符串作为键和值. (shelve 模块可以处理任何类型的值)

10.2. anydbm 模块

anydbm 模块为简单数据库驱动提供了统一标准的接口.

当第一次被导入的时候, anydbm 模块会自动寻找一个合适的数据库驱动, 按照 dbhash , gdbm , dbm , 或 dumbdbm 的顺序尝试. 如果没有找到任何模块, 它将引发一个 *ImportError* 异常.

open 函数用于打开或创建一个数据库(使用导入时找到的数据库驱动), 如 Example 10-1 所示.

10.2.0.1. Example 10-1. 使用 anydbm 模块

```
File: anydbm-example-1.py



import anydbm



db = anydbm.open("database", "c")

db["1"] = "one"

db["2"] = "two"

db["3"] = "three"

db.close()



db = anydbm.open("database", "r")

for key in db.keys():

    print repr(key), repr(db[key])
```

```
'2' 'two'
```

```
'3' 'three'

'1' 'one'
```

10.3. whichdb 模块

whichdb 模块可以判断给定数据库文件的格式, 如 Example 10-2 所示.

10.3.0.1. Example 10-2. 使用 whichdb 模块

```
File: whichdb-example-1.py



import whichdb



filename = "database"



result = whichdb.whichdb(filename)
```

```
if result:
```

```
    print "file created by", result

    handler = _ _import_ _(result)
```

```
        db = handler.open(filename, "r")

        print db.keys()

    else:

        # cannot identify data base

        if result is None:

            print "cannot read database file", filename

        else:

            print "cannot identify database file",
    filename

        db = None
```

这个例子中使用了 `_ _import_ _` 函数来导入对应模块(还记得我们在第一章的例子么?).

10.4. shelve 模块

`shelve` 模块使用数据库驱动实现了字典对象的持久保存. `shelve` 对象使用字符串作为键, 但值可以是任意类型, 所有可以被 `pickle` 模块处理的对象都可以作为它的值. 如 Example 10-3 所示.

10.4.0.1. Example 10-3. 使用 shelve 模块

```
    File: shelve-example-1.py
```

```
import shelve



db = shelve.open("database", "c")

db["one"] = 1

db["two"] = 2

db["three"] = 3

db.close()



db = shelve.open("database", "r")

for key in db.keys():

    print repr(key), repr(db[key])
```

```
'one' 1
```

```
'three' 3

'two' 2
```

Example 10-4 展示了如何使用 shelve 处理给定的数据库驱动.

10.4.0.2. Example 10-4. 使用 shelve 模块处理给定数据库

```
File: shelve-example-3.py



import shelve

import gdbm



def gdbm_shelve(filename, flag="c"):

    return shelve.Shelf(gdbm.open(filename, flag))



db = gdbm_shelve("dbfile")
```

10.5. dbhash 模块

(可选) dbhash 模块为 bsddb 数据库驱动提供了一个 dbm 兼容的接口. 如 Example 10-5 所示.

10.5.0.1. Example 10-5. 使用 dbhash 模块

```
File: dbhash-example-1.py



import dbhash
```

```
db = dbhash.open("dbhash", "c")

db["one"] = "the foot"

db["two"] = "the shoulder"

db["three"] = "the other foot"

db["four"] = "the bridge of the nose"

db["five"] = "the naughty bits"

db["six"] = "just above the elbow"

db["seven"] = "two inches to the right of a very
naughty bit indeed"

db["eight"] = "the kneecap"

db.close()
```

```

```

```
db = dbhash.open("dbhash", "r")

for key in db.keys():

    print repr(key), repr(db[key])
```

10.6. dbm 模块

(可选) dbm 模块提供了一个到 dbm 数据库驱动的接口(在许多 Unix 平台上都可用). 如 Example 10-6 所示.

10.6.0.1. Example 10-6. 使用 dbm 模块

```
File: dbm-example-1.py



import dbm



db = dbm.open("dbm", "c")

db["first"] = "bruce"

db["second"] = "bruce"

db["third"] = "bruce"
```

```
db["fourth"] = "bruce"
```

```
db["fifth"] = "michael"

db["fifth"] = "bruce" # overwrite

db.close()



db = dbm.open("dbm", "r")

for key in db.keys():
```

```
    print repr(key), repr(db[key])



'first' 'bruce'

'second' 'bruce'

'fourth' 'bruce'

'third' 'bruce'

'fifth' 'bruce'
```

10.7. dumbdbm 模块

dumbdbm 模块是一个简单的数据库实现，与 dbm 一类相似，但使用纯 Python 实现. 它使用两个文件：一个二进制文件（*.dat*）用于储存数据，一个文本文件（*.dir*）用于数据描述.

10.7.0.1. Example 10-7. 使用 dumbdbm 模块

```
File: dumbdbm-example-1.py



import dumbdbm



db = dumbdbm.open("dumbdbm", "c")
```

```
db["first"] = "fear"

db["second"] = "surprise"

db["third"] = "ruthless efficiency"

db["fourth"] = "an almost fanatical devotion to the
Pope"

db["fifth"] = "nice red uniforms"

db.close()



db = dumbdbm.open("dumbdbm", "r")

for key in db.keys():

    print repr(key), repr(db[key])
```

```
'first' 'fear'

'third' 'ruthless efficiency'

'fifth' 'nice red uniforms'

'second' 'surprise'

'fourth' 'an almost fanatical devotion to the Pope'
```

10.8. gdbm 模块

(可选) gdbm 模块提供了到 GNU dbm 数据驱动的接口, 如 Example 10-8 所示.

10.8.0.1. Example 10-8. 使用 gdbm 模块

```
File: gdbm-example-1.py



import gdbm



db = gdbm.open("gdbm", "c")
```

```
db["1"] = "call"
```

```
db["2"] = "the"

db["3"] = "next"

db["4"] = "defendant"

db.close()




db = gdbm.open("gdbm", "r")
```

```
keys = db.keys()

keys.sort()

for key in keys:

    print db[key],



call the next defendant
```

11. 工具和实用程序

标准库中有一些模块既可用作模块又可以作为命令行实用程序.

11.1. dis 模块

`dis` 模块是 Python 的反汇编器. 它可以把字节码转换为更容易让人看懂的格式.

你可以从命令行调用反汇编器. 它会编译给定的脚本并把反汇编后的字节代码输出到终端上:

```
$ dis.py hello.py



          0 SET_LINENO               0
```

```
          3 SET_LINENO               1

          6 LOAD_CONST               0 ('hello
again, and welcome to the show')

          9 PRINT_ITEM

         10 PRINT_NEWLINE

         11 LOAD_CONST               1 (None)

         14 RETURN_VALUE
```

当然 dis 也可以作为模块使用. dis 函数接受一个类, 方法, 函数, 或者 code 对象作为单个参数. 如 Example 11-1 所示.

11.1.0.1. Example 11-1. 使用 dis 模块

```
File: dis-example-1.py



import dis



def procedure():

    print 'hello'
```

```
dis.dis(procedure)


          0 SET_LINENO          3


          3 SET_LINENO          4
          6 LOAD_CONST          1 ('hello')
          9 PRINT_ITEM
         10 PRINT_NEWLINE
         11 LOAD_CONST          0 (None)
         14 RETURN_VALUE
```

11.2. pdb 模块

pdb 模块是标准 Python 调试器(debugger). 它基于 bdb 调试器框架.

你可以从命令行调用调试器 (键入 n 或 进入下一行代码, 键入 help 获得可用命令列表):

```
$ pdb.py hello.py

> hello.py(0)?()
```

```
(Pdb) n
> hello.py()
(Pdb) n
hello again, and welcome to the show
--Return--
> hello.py(1)?()->None
(Pdb)
```

Example 11-2 展示了如何从程序中启动调试器.

11.2.0.1. Example 11-2. 使用 pdb 模块

```
File: pdb-example-1.py



import pdb



def test(n):
    j = 0
    for i in range(n):
        j = j + i
```

```
    return n



db = pdb.Pdb()

db.runcall(test, 1)



> pdb-example-1.py(3)test()

-> def test(n):

(Pdb) s

> pdb-example-1.py(4)test()
```

```
-> j = 0
```

```
(Pdb) s

> pdb-example-1.py(5)test()

-> for i in range(n):

...
```

11.3. bdb 模块

bdb 模块为提供了一个调试器框架. 你可以使用它来创建自定义的调试器, 如 Example 11-3 所示.

你需要做的只是继承 *Bdb* 类, 覆盖它的 user 方法(在每次调试器停止的时候被调用). 使用各种各样的 set 方法可以控制调试器.

11.3.0.1. Example 11-3. 使用 bdb 模块

```
File: bdb-example-1.py



import bdb

import time



def spam(n):
```

```
    j = 0
```

```
    for i in range(n):

        j = j + i

    return n



def egg(n):

    spam(n)

    spam(n)
```

```
    spam(n)

    spam(n)



def test(n):

    egg(n)



class myDebugger(bdb.Bdb):



    run = 0
```

```

```

```
    def user_call(self, frame, args):

        name = frame.f_code.co_name or "<unknown>"

        print "call", name, args

        self.set_continue() # continue



    def user_line(self, frame):

        if self.run:
```

```
            self.run = 0

            self.set_trace() # start tracing

        else:

            # arrived at breakpoint

            name = frame.f_code.co_name or
"<unknown>"

            filename =
self.canonic(frame.f_code.co_filename)

            print "break at", filename,
frame.f_lineno, "in", name

        print "continue..."
```

```
        self.set_continue() # continue to next
breakpoint
```

```
    def user_return(self, frame, value):

        name = frame.f_code.co_name or "<unknown>"

        print "return from", name, value

        print "continue..."

        self.set_continue() # continue
```

```
    def user_exception(self, frame, exception):
        name = frame.f_code.co_name or "<unknown>"
        print "exception in", name, exception
        print "continue..."
        self.set_continue() # continue

db = myDebugger()
db.run = 1
db.set_break("bdb-example-1.py", 7)
```

```
db.runcall(test, 1)
```

```
continue...
call egg None
call spam None
break at C:\ematter\librarybook\bdb-example-1.py 7
in spam
```

```
continue...

call spam None

break at C:\ematter\librarybook\bdb-example-1.py 7
in spam

continue...

call spam None

break at C:\ematter\librarybook\bdb-example-1.py 7
in spam

continue...

call spam None

break at C:\ematter\librarybook\bdb-example-1.py 7
in spam

continue...
```

11.4. profile 模块

profile 模块是标准 Python 分析器.
和反汇编器, 调试器相同, 你可以从命令行调用分析器:

```
$ profile.py hello.py
```

```
hello again, and welcome to the show



         3 function calls in 0.785 CPU seconds



  Ordered by: standard name



  ncalls  tottime  percall  cumtime  percall
filename:lineno(function)

       1  0.001    0.001    0.002    0.002
<string>:1(?)

       1  0.001    0.001    0.001    0.001
hello.py:1(?)

       1  0.783    0.783    0.785    0.785
profile:0(execfile('hello.py'))
```

```
       0  0.000             0.000
profile:0(profiler)
```

如 Example 11-4 所示, 我们还可以从程序中调用 profile 来对程序性能做分析.

11.4.0.1. Example 11-4. U 使用 profile 模块

```
File: profile-example-1.py
```

```
import profile


def func1():
    for i in range(1000):
        pass


def func2():
    for i in range(1000):
        func1()


profile.run("func2()")
```

```
         1003 function calls in 2.380 CPU seconds


   Ordered by: standard name
```

```
  ncalls  tottime  percall  cumtime  percall
filename:lineno(function)

       1  0.000    0.000    2.040    2.040
<string>:1(?)

    1000  1.950    0.002    1.950    0.002
profile-example-1.py:3(func1)

       1  0.090    0.090    2.040    2.040
profile-example-1.py:7(func2)

       1  0.340    0.340    2.380    2.380
profile:0(func2())

       0  0.000             0.000
profile:0(profiler)
```

你可以使用 pstats 模块来修改结果报告的形式.

11.5. pstats 模块

pstats 模块用于分析 Python 分析器收集的数据. 如 Example 11-5 所示.

11.5.0.1. Example 11-5. 使用 pstats 模块

```
File: pstats-example-1.py



import pstats

import profile
```

```
def func1():
    for i in range(1000):
        pass

def func2():
    for i in range(1000):
        func1()

p = profile.Profile()
p.run("func2()")
```

```
s = pstats.Stats(p)
```

```
s.sort_stats("time", "name").print_stats()
```

```
         1003 function calls in 1.574 CPU seconds
```

```
   Ordered by: internal time, function name



   ncalls  tottime  percall  cumtime  percall
filename:lineno(function)

     1000    1.522    0.002    1.522    0.002
pstats-example-1.py:4(func1)

        1    0.051    0.051    1.573    1.573
pstats-example-1.py:8(func2)

        1    0.001    0.001    1.574    1.574
profile:0(func2())

        1    0.000    0.000    1.573    1.573
<string>:1(?)

        0    0.000             0.000
profile:0(profiler)
```

11.6. tabnanny 模块

(2.0 新增) `tabnanny` 模块用于检查 Python 源文件中的含糊的缩进. 当文件混合了 tab 和空格两种缩进时候, nanny (保姆)会立即给出提示.

在下边使用的 `badtabs.py` 文件中, if 语句后的第一行使用 4 个空格和 1 个 tab . 第二行只使用了空格.

```
$ tabnanny.py -v samples/badtabs.py

';samples/badtabs.py': *** Line 3: trouble in tab
city! ***
```

```
offending line:         print "world"



indent not equal e.g. at tab sizes 1, 2, 3, 5, 6, 7,
9
```

因为 Python 解释器把 tab 作为 8 个空格来处理, 所以这个脚本可以正常运行. 在所有符合代码标准(一个 tab 为 8 个空格)的编辑器中它也会正常显示. 当然, 这些都骗不过 nanny .

Example 11-6 展示了如何在你自己的程序中使用 tabnanny .

11.6.0.1. Example 11-6. 使用 tabnanny 模块

```
File: tabnanny-example-1.py



import tabnanny
```

```
FILE = "samples/badtabs.py"
```

```


file = open(FILE)

for line in file.readlines():

    print repr(line)
```

```
# let tabnanny look at it

tabnanny.check(FILE)



'if 1:\012'

'    \011print "hello"\012'

'        print "world"\012'

samples/badtabs.py 3 '        print "world"'\012'
```

将 sys.stdout 重定向到一个 StringIO 对象就可以捕获输出.

12. 其他模块

12.1. 概览

本章介绍了一些平台相关的模块. 重点放在了适用于整个平台家族的模块上. (比如 Unix , Windows 家族)

12.2. fcntl 模块

(只用于 Unix) fcntl 模块为 Unix 上的 ioctl 和 fcntl 函数提供了一个接口. 它们用于文件句柄和 I/O 设备句柄的 "out of band" 操作, 包括读取扩展属性, 控制阻塞. 更改终端行为等等. (out of band management: 指使用分离的渠道进行设备管理. 这使系统管理员能在机器关机的时候对服务器, 网络进行监视和管理. 出处: http://en.wikipedia.org/wiki/Out-of-band_management)

关于如何在平台上使用这些函数，请查阅对应的 Unix man 手册.

该模块同时提供了 Unix 文件锁定机制的接口. Example 12-1 展示了如何使用 `flock` 函数，更新文件时为文件设置一个 *advisory lock* .

输出结果是由同时运行 3 个副本得到的. 像这样(都在一句命令行里):

```
python fcntl-example-1.py& python
fcntl-example-1.py& python fcntl-example-1.py&
```

如果你注释掉对 `flock` 的调用，那么 counter 文件不会正确地更新.

12.2.0.1. Example 12-1. Using the fcntl Module

```
File: fcntl-example-1.py



import fcntl, FCNTL

import os, time
```

```

```

```
FILE = "counter.txt"



if not os.path.exists(FILE):

    # create the counter file if it doesn't exist

     # 创建 counter 文件

    file = open(FILE, "w")
```

```
        file.write("0")
        file.close()


for i in range(20):
    # increment the counter
    file = open(FILE, "r+")
    fcntl.flock(file.fileno(), FCNTL.LOCK_EX)
    counter = int(file.readline()) + 1
    file.seek(0)
    file.write(str(counter))
```

```
    file.close() # unlocks the file
```

```
    print os.getpid(), "=>", counter
    time.sleep(0.1)


30940 => 1
30942 => 2
30941 => 3
```

```
30940 => 4

30941 => 5

30942 => 6
```

12.3. pwd 模块

(只用于 Unix) pwd 提供了一个到 Unix 密码/password "数据库 "(/etc/passwd 以及相关文件)的接口. 这个数据库(一般是一个纯文本文件)包含本地机器用户账户的信息. 如 Example 12-2 所示.

12.3.0.1. Example 12-2. 使用 pwd 模块

```
File: pwd-example-1.py
```

```
import pwd
```

```
import os



print pwd.getpwuid(os.getgid())

print pwd.getpwnam("root")
```

```
('effbot', 'dsWjk8', 4711, 4711, 'eff-bot',
'/home/effbot', '/bin/bosh')

('root', 'hs2giiw', 0, 0, 'root', '/root',
'/bin/bash')
```

getpwall 函数返回一个包含所有可用用户数据库入口的列表. 你可以使用它搜索一个用户.

当需要查询很多名称的时候, 你可以使用 getpwall 来预加载一个字典, 如 Example 12-3 所示.

12.3.0.2. Example 12-3. 使用 pwd 模块

```
File: pwd-example-2.py



import pwd

import os
```

```

```

```
# preload password dictionary

_pwd = {}

for info in pwd.getpwall():

    _pwd[info[0]] = _pwd[info[2]] = info
```

```
def userinfo(uid):

    # name or uid integer

    return _pwd[uid]



print userinfo(os.getuid())

print userinfo("root")



('effbot', 'dsWjk8', 4711, 4711, 'eff-bot',
'/home/effbot', '/bin/bosh')

('root', 'hs2giiw', 0, 0, 'root', '/root',
'/bin/bash')
```

12.4. grp 模块

(只用于 Unix) grp 模块提供了一个到 Unix 用户组/group (/etc/group) 数据库的接口. getgrgid 函数返回给定用户组 id 的相关数据(参见 Example 12-4), getgrnam 返回给定用户组名称的相关数据.

12.4.0.1. Example 12-4. 使用 grp 模块

```
File: grp-example-1.py



import grp
```

```
import os



print grp.getgrgid(os.getgid())

print grp.getgrnam("wheel")



('effbot', '', 4711, ['effbot'])

('wheel', '', 10, ['root', 'effbot', 'gorbot',
'timbot'])
```

`getgrall` 函数返回包含所有可用用户组数据库入口的列表.

如果需要执行很多用户组查询, 你可以使用 `getgrall` 来把当前所有的用户组复制到一个字典里, 这可以节省一些时间. Example 12-5 中的 `groupinfo` 函数返回一个用户组 id (int)或是一个用户组名称(str) 的信息.

12.4.0.2. Example 12-5. 使用 grp 模块缓存用户组信息

```
File: grp-example-2.py



import grp

import os



# preload password dictionary
```

```
_grp = {}

for info in grp.getgrall():

    _grp[info[0]] = _grp[info[2]] = info



def groupinfo(gid):

    # name or gid integer

    return _grp[gid]
```

```
print groupinfo(os.getgid())
```

```
print groupinfo("wheel")
```

```
('effbot', '', 4711, ['effbot'])

('wheel', '', 10, ['root', 'effbot', 'gorbot',
'timbot'])
```

12.5. nis 模块

(̣ Unix , ω) nis ģ ŝ NIS (Network Information Services , Ų , ̨) Ľ□, Example 12-6 □. ک□õ NIS □ л .

12.5.0.1. Example 12-6. □ nis ģ

```
File: nis-example-1.py



import nis

import string
```

```
print nis.cat("ypservers")
```

```
print string.split(nis.match("bacon",
"hosts.byname"))
```

```
{'bacon.spam.egg': 'bacon.spam.egg'}

['194.18.155.250', 'bacon.spam.egg', 'bacon',
'spam-010']
```

12.6. curses 模块

(ˌ���� Unix ��ω) curses ģ���ş□��ı��¸��ŭ□ ��ǰŁ���, ��□����h□ˌ������ŭ□ķ���. ��

Example 12-7

12-7 ��□.

12.6.0.1. Example 12-7. □�� curses ģ��

```
File: curses-example-1.py



import curses



text = [

    "a very simple curses demo",

    "",

    "(press any key to exit)"

]



# connect to the screen

# ���ų���Ļ
```

```
screen = curses.initscr()



# setup keyboard

# ◆◆◆ü◆◆◆

curses.noecho() # no keyboard echo

curses.cbreak() # don't wait for newline



# screen size

# ◆◆Ļ◆□◆
```

```
rows, columns = screen.getmaxyx()
```

```


# draw a border around the screen

# ◆◆h◆◆◆□◆

screen.border()



# display centered text
```

```
# ��□����

y = (rows - len(text)) / 2



for line in text:

    screen.addstr(y, (columns-len(line))/2, line)

    y = y + 1



screen.getch()



curses.endwin()
```

12.7. termios 模块

(只用于 Unix , 可选) termios 为 Unix 的终端控制设备提供了一个接口. 它可用于控制终端通讯端口的大多方面.

Example 12-8 中, 该模块临时关闭了键盘回显(由第三个标志域的 ECHO 标志控制).

12.7.0.1. Example 12-8. 使用 termios 模块

```
File: termios-example-1.py
```

```
import termios, TERMIOS

import sys



fileno = sys.stdin.fileno()



attr = termios.tcgetattr(fileno)

orig = attr[:]
```

```
print "attr =>", attr[:4] # flags
```

```


# disable echo flag

attr[3] = attr[3] & ~TERMIOS.ECHO



try:

    termios.tcsetattr(fileno, TERMIOS.TCSADRAIN,
attr)
```

```
    message = raw_input("enter secret message: ")

    print

finally:

    # restore terminal settings

    termios.tcsetattr(fileno, TERMIOS.TCSADRAIN,
orig)



print "secret =>", repr(message)



attr => [1280, 5, 189, 35387]

enter secret message:
```

```
secret => 'and now for something completely
different'
```

12.8. tty 模块

(只用于 Unix) `tty` 模块包含一些用于处理 tty 设备的工具函数. Example 12-9 将终端窗口切换为 "raw" 模式.

12.8.0.1. Example 12-9. 使用 tty 模块

```
File: tty-example-1.py



import tty

import os, sys



fileno = sys.stdin.fileno()



tty.setraw(fileno)

print raw_input("raw input: ")



tty.setcbreak(fileno)
```

```
print raw_input("cbreak input: ")
```

```


os.system("stty sane") # ...



raw input: this is raw input
```

```
cbreak input: this is cbreak input
```

12.9. resource 模块

(只用于 Unix , 可选) resource 模块用于查询或修改当前系统资源限制设置. Example 12-10 展示了如何执行查询操作, Example 12-11 展示了如何执行修改操作.

12.9.0.1. Example 12-10. 使用 resource 模块查询当前设置

```
File: resource-example-1.py



import resource



print "usage stats", "=>",
resource.getrusage(resource.RUSAGE_SELF)
```

```
print "max cpu", "=>",
resource.getrlimit(resource.RLIMIT_CPU)
```

```
print "max data", "=>",
resource.getrlimit(resource.RLIMIT_DATA)

print "max processes", "=>",
resource.getrlimit(resource.RLIMIT_NPROC)

print "page size", "=>", resource.getpagesize()
```

```
usage stats => (0.03, 0.02, 0, 0, 0, 0, 75, 168, 0,
0, 0, 0, 0, 0, 0, 0)

max cpu => (2147483647, 2147483647)

max data => (2147483647, 2147483647)

max processes => (256, 256)

page size => 4096
```

12.9.0.2. Example 12-11. 使用 resource 模块限制资源

```
File: resource-example-2.py



import resource



resource.setrlimit(resource.RLIMIT_CPU, (0, 1))



# pretend we're busy

for i in range(1000):

    for j in range(1000):
```

```
        for k in range(1000):

            pass



CPU time limit exceeded
```

12.10. syslog 模块

(只用于 Unix 可选) syslog 模块用于向系统日志设备发送信息(syslogd). 这些信息如何处理依不同的系统而定, 通常会被记录在一个 log 文件中, 例如 */var/log/messages* , */var/adm/syslog* , 或者其他类似处理. (如果你找不到这个文件, 请联系你的系统管理员). Example 12-12 展示了该模块的使用.

12.10.0.1. Example 12-12. 使用 syslog 模块

```
File: syslog-example-1.py



import syslog
```

```
import sys
```

```


syslog.openlog(sys.argv[0])


```

```
syslog.syslog(syslog.LOG_NOTICE, "a log notice")

syslog.syslog(syslog.LOG_NOTICE, "another log
notice: %s" % "watch out!")



syslog.closelog()
```

12.11. msvcrt 模块

(只用于 Windows/DOS) `msvcrt` 模块用于访问 Microsoft Visual C/C++ Runtime Library (MSVCRT) 中函数的方法.

Example 12-13 展示了 `getch` 函数, 它用于从命令行读取一次按键操作.

12.11.0.1. Example 12-13. 使用 msvcrt 模块获得按键值

```
File: msvcrt-example-1.py



import msvcrt
```

```

```

```
print "press 'escape' to quit..."



while 1:
```

```
    char = msvcrt.getch()
    if char == chr(27):
        break
    print char,
    if char == chr(13):
        print
```

```
press 'escape' to quit...
h e l l o
```

kbhit 函数在按键时返回(这样的捕获操作不会让 getch 阻塞), 如 Example 12-14 所示.

12.11.0.2. Example 12-14. 使用 msvcrt 模块接受键盘输入

```
File: msvcrt-example-2.py
```

```
import msvcrt
import time
```

```
print "press SPACE to enter the serial number"



while not msvcrt.kbhit() or msvcrt.getch() != " ":

    # do something else

    print ".",

    time.sleep(0.1)



print



# clear the keyboard buffer

# 清除键盘缓冲区

while msvcrt.kbhit():

    msvcrt.getch()
```

```
serial = raw_input("enter your serial number: ")



print "serial number is", serial
```

```
press SPACE to enter the serial number

. . . . . .

enter your serial number: 10

serial number is 10
```

译注：某翻译在这里评注道：我能在 cmd 下运行. 用别的 IDLE 要不然卡住，要不然接受不了键盘输入. 原因未知. 这是因为 IDLE 启动两个 python 线程，使用 socket 发送数据，获得程序返回的.

locking 函数实现了 Windows 下的跨进程文件锁定，如 Example 12-15 所示.

12.11.0.3. Example 12-15. 使用 msvcrt 模块锁定文件

```
File: msvcrt-example-3.py



import msvcrt

import os
```

```

```

```
LK_UNLCK = 0 # unlock the file region 解锁区域

LK_LOCK = 1 # lock the file region 锁定文件区域

LK_NBLCK = 2 # non-blocking lock 非阻塞文件锁
```

```
LK_RLCK = 3 # lock for writing 为写入文件提供锁定

LK_NBRLCK = 4 # non-blocking lock for writing 为写入文件提供的非阻塞锁定



FILE = "counter.txt"



if not os.path.exists(FILE):

    file = open(FILE, "w")

    file.write("0")

    file.close()



for i in range(20):

    file = open(FILE, "r+")

    # look from current position (0) to end of file

    msvcrt.locking(file.fileno(), LK_LOCK,
os.path.getsize(FILE))

    counter = int(file.readline()) + 1

    file.seek(0)
```

```
    file.write(str(counter))

    file.close() # unlocks the file

    print os.getpid(), "=>", counter

    time.sleep(0.1)



208 => 21

208 => 22

208 => 23

208 => 24

208 => 25

208 => 26
```

12.12. nt 模块

(非直接使用模块, 只用于 Windows) nt 模块是 os 模块在 Windows 平台下调用的执行模块. 几乎没有任何原因直接使用这个模块, 请使用 os 模块替代. Example 12-16 展示了它的使用.

12.12.0.1. Example 12-16. 使用 nt 模块

```
File: nt-example-1.py
```

```
import nt



# in real life, use os.listdir and os.stat instead!

for file in nt.listdir("."):

    print file, nt.stat(file)[6]



aifc-example-1.py 314

anydbm-example-1.py 259

array-example-1.py 48
```

12.13. _winreg 模块

(只用于 Windows，2.0 中新增) _winreg 模块提供了访问 Windows 注册表数据库的一个基本接口. Example 12-17 展示了它的使用.

12.13.0.1. Example 12-17. 使用 _winreg 模块

```
File: winreg-example-1.py



import _winreg
```

```
explorer = _winreg.OpenKey(
    _winreg.HKEY_CURRENT_USER,
"Software\\Microsoft\\Windows\CurrentVersion\\Explorer"
    )


# list values owned by this registry key
# 列出该注册表键下的所有值
try:
    i = 0
    while 1:
```

```
      name, value, type=
_winreg.EnumValue(explorer, i)
```

```
      print repr(name),
      i += 1
except WindowsError:
    print
```

```
value, type = _winreg.QueryValueEx(explorer, "Logon
User Name")




print


print "user is", repr(value)
```

```
'Logon User Name' 'CleanShutdown' 'ShellState'
'Shutdown Setting'

'Reason Setting' 'FaultCount' 'FaultTime'
'IconUnderline'...
```

```
user is u'Effbot'
```

12.14. posix 模块

(非直接使用模块, 只用于 Unix/POSIX) posix 模块是 os 模块在 Unix 及其他 POSIX 系统下使用的实现模块. 一般只需要通过 os 模块访问它即可. 如 Example 12-18 所示.

12.14.0.1. Example 12-18. 使用 posix 模块

```
File: posix-example-1.py



import posix



for file in posix.listdir("."):

    print file, posix.stat(file)[6]



aifc-example-1.py 314

anydbm-example-1.py 259

array-example-1.py 48
```

13. 执行支持模块

就是其他模块中用到的模块.

13.1. dospath 模块

dospath 模块(参见 Example 13-1)提供了 DOS 平台下的 os.path 功能. 你可以使用它在其他平台处理 DOS 路径.

13.1.0.1. Example 13-1. 使用 dospath 模块

File: dospath-example-1.py

```
import dospath

file = "/my/little/pony"

print "isabs", "=>", dospath.isabs(file)
print "dirname", "=>", dospath.dirname(file)
print "basename", "=>", dospath.basename(file)
print "normpath", "=>", dospath.normpath(file)
print "split", "=>", dospath.split(file)
print "join", "=>", dospath.join(file, "zorba")
```

```
isabs => 1
dirname => /my/little
basename => pony
normpath => \my\little\pony
```

```
split => ('/my/little', 'pony')

join => /my/little/pony\zorba
```

注意 Python 的 DOS 支持可以使用斜杠和反斜杠作为目录分隔符.

13.2. macpath 模块

macpath 模块(参见 Example 13-2)提供了 Macintosh 平台下的 os.path 功能. 你也可以使用它在其他平台处理 Macintosh 路径.

13.2.0.1. Example 13-2. 使用 macpath 模块

```
File: macpath-example-1.py



import macpath



```

```
file = "my:little:pony"
```

```


print "isabs", "=>", macpath.isabs(file)

print "dirname", "=>", macpath.dirname(file)

print "basename", "=>", macpath.basename(file)

print "normpath", "=>", macpath.normpath(file)
```

```
print "split", "=>", macpath.split(file)

print "join", "=>", macpath.join(file, "zorba")



isabs => 1

dirname => my:little

basename => pony

normpath => my:little:pony

split => ('my:little', 'pony')

join => my:little:pony:zorba
```

13.3. ntpath 模块

ntpath 模块(参见 Example 13-3)提供了 Windows 平台下的 os.path 功能. 你也可以使用它在其他平台处理 Windows 路径.

13.3.0.1. Example 13-3. 使用 ntpath 模块

```
File: ntpath-example-1.py



import ntpath
```

```
file = "/my/little/pony"



print "isabs", "=>", ntpath.isabs(file)

print "dirname", "=>", ntpath.dirname(file)

print "basename", "=>", ntpath.basename(file)

print "normpath", "=>", ntpath.normpath(file)

print "split", "=>", ntpath.split(file)

print "join", "=>", ntpath.join(file, "zorba")
```

```
isabs => 1

dirname => /my/little
```

```
basename => pony
```

```


normpath => \my\little\pony

split => ('/my/little', 'pony')

join => /my/little/pony\zorba
```

注意该模块可以同时使用斜杠和反斜杠作为目录分隔符.

13.4. posixpath 模块

posixpath 模块(参见 Example 13-4)提供了 Unix 和其他 POSIX 兼容平台下的 os.path 功能. 你也可以使用它在其他平台处理 POSIX 路径. 另外, 它也可以处理 URL .

13.4.0.1. Example 13-4. 使用 posixpath 模块

```
File: posixpath-example-1.py



import posixpath



file = "/my/little/pony"
```

```
print "isabs", "=>", posixpath.isabs(file)
```

```
print "dirname", "=>", posixpath.dirname(file)

print "basename", "=>", posixpath.basename(file)

print "normpath", "=>", posixpath.normpath(file)

print "split", "=>", posixpath.split(file)

print "join", "=>", posixpath.join(file, "zorba")
```

```
isabs => 1

dirname => /my/little

basename => pony

normpath => /my/little/pony

split => ('/my/little', 'pony')

join => /my/little/pony/zorba
```

13.5. strop 模块

(已废弃) strop 为 string 模块中的大多函数提供了底层 C 语言实现. string 模块会自动调用它, 所以一般你不需要直接使用它.

不过在导入 Python 模块之前处理路径的时候你可能会用到它. 如 Example 13-5 所示.

13.5.0.1. Example 13-5. 使用 strop 模块

```
File: strop-example-1.py



import strop

import sys


```

```
# assuming we have an executable named
"../executable", add a

# directory named "../executable-extra" to the path



if strop.lower(sys.executable)[-4:] == ".exe":

    extra = sys.executable[:-4] # windows

else:

    extra = sys.executable



sys.path.insert(0, extra + "-extra")
```

```
import mymodule
```

在 Python 2.0 及以后版本中，你应该使用字符串方法代替 `strop`，例如在上边的代码中. 使用 `"sys.executable.lower()"` 替换 `"strop.lower(sys.executable)"`.

13.6. imp 模块

`imp` 模块包含的函数可以用于实现自定义的 import 行为. Example 13-6 重载了 `import` 语句，实现了对模块来源的记录功能.

13.6.0.1. Example 13-6. 使用 imp 模块

File: imp-example-1.py

```
import imp

import sys



def my_import(name, globals=None, locals=None,
fromlist=None):

    try:

        module = sys.modules[name] # already
imported?

    except KeyError:

        file, pathname, description =
imp.find_module(name)

        print "import", name, "from", pathname,
description

        module = imp.load_module(name, file,
pathname, description)

    return module



import _ _builtin_ _
```

```
_ _builtin_ _._ _import_ _ = my_import



import xmllib



import xmllib from /python/lib/xmllib.py ('.py',
'r', 1)

import re from /python/lib/re.py ('.py', 'r', 1)



import sre from /python/lib/sre.py ('.py', 'r', 1)
```

```
import sre_compile from /python/lib/sre_compile.py
('.py', 'r', 1)
```

```
import _sre from /python/_sre.pyd ('.pyd', 'rb', 3)
```

注意这里的导入功能不支持包. 具体实现请参阅 knee 模块的源代码.

13.7. new 模块

new 模块是一个底层的模块, 你可以使用它来创建不同的内建对象, 例如类对象, 函数对象, 以及其他由 Python 运行时系统创建的类型. Example 13-7 展示了该模块的使用.

如果你使用的是 1.5.2 版本 , 那么你有可能需要重新编译 Python 来使用这个模块, 在默认情况下并不是所有平台都有这个模块. 在 2.0 及以后版本中, 不需要这么做.

13.7.0.1. Example 13-7. 使用 new 模块

```
File: new-example-1.py



import new



class Sample:



    a = "default"
```

```

```

```
    def __init__(self):

        self.a = "initialised"



    def __repr__(self):

        return self.a



#

# create instances
```

```
a = Sample()

print "normal", "=>", a



b = new.instance(Sample, {})

print "new.instance", "=>", b



b._ _init_ _()
```

```
print "after _ _init_ _", "=>", b
```

```
c = new.instance(Sample, {"a": "assigned"})

print "new.instance w. dictionary", "=>", c



normal => initialised

new.instance => default

after _ _init_ _ => initialised
```

```
new.instance w. dictionary => assigned
```

13.8. pre 模块

(已废弃) pre 模块是 1.5.2 中 re 模块调用的实现功能模块. 在当前版本中已废弃. Example 13-8 展示了它的使用.

13.8.0.1. Example 13-8. 使用 pre 模块

```
File: pre-example-1.py



import pre
```

```

```

```
p = pre.compile("[Python]+")



print p.findall("Python is not that bad")



['Python', 'not', 'th', 't']
```

13.9. sre 模块

(功能实现模块, 已声明不支持) sre 模块是 re 模块的底层实现. 一般没必要直接使用它, 而且以后版本将不会支持它. Example 13-9 展示了它的使用.

13.9.0.1. Example 13-9. 使用 sre 模块

```
File: sre-example-1.py



import sre



text = "The Bookshop Sketch"
```

```
# a single character
```

```
m = sre.match(".", text)

if m: print repr("."), "=>", repr(m.group(0))



# and so on, for all 're' examples...
```

```
'.' => 'T'
```

13.10. py_compile 模块

py_compile 模块用于将 Python 模块编译为字节代码. 它和 Python 的 import 语句行为类似, 不过它接受文件名而不是模块名作为参数. 使用方法如 Example 13-10 所示.

13.10.0.1. Example 13-10. 使用 py_compile 模块

```
File: py-compile-example-1.py



import py_compile



# explicitly compile this module

py_compile.compile("py-compile-example-1.py")
```

compileall 模块可以把一个目录树下的所有 Python 文件编译为字节代码.

13.11. compileall 模块

compileall 模块用于将给定目录下(以及 Python path)的所有 Python 脚本编译为字节代码. 它也可以作为可执行脚本使用(在 Unix 系统下, Python 安装时会自动调用执行它). 用法参见 Example 13-11 .

13.11.0.1. Example 13-11. 使用 compileall 模块编译目录中的所有脚本

```
File: compileall-example-1.py
```

```
import compileall

print "This may take a while!"

compileall.compile_dir(".", force=1)
```

```
This may take a while!
Listing . ...
Compiling .\SimpleAsyncHTTP.py ...
Compiling .\aifc-example-1.py ...
Compiling .\anydbm-example-1.py ...
...
```

13.12. ihooks 模块

ihooks 模块为替换导入提供了一个框架. 这允许多个导入机制共存. 使用方法参见 Example 13-12 .

13.12.0.1. Example 13-12. 使用 ihooks 模块

```
File: ihooks-example-1.py



import ihooks, imp, os



def import_from(filename):

    "Import module from a named file"



    loader = ihooks.BasicModuleLoader()

    path, file = os.path.split(filename)

    name, ext  = os.path.splitext(file)

    m = loader.find_module_in_dir(name, path)

    if not m:

        raise ImportError, name

    m = loader.load_module(name, m)

    return m
```

```
colorsys = import_from("/python/lib/colorsys.py")



print colorsys



<module 'colorsys' from '/python/lib/colorsys.py'>
```

13.13. linecache 模块

linecache 模块用于从模块源文件中读取代码. 它会缓存最近访问的模块(整个源文件). 如 Example 13-13 .

13.13.0.1. Example 13-13. 使用 linecache 模块

```
File: linecache-example-1.py



import linecache



print linecache.getline("linecache-example-1.py",
5)



print linecache.getline("linecache-example-1.py",
5)
```

traceback 模块使用这个模块实现了对导入操作的跟踪.

13.14. macurl2path 模块

(功能实现模块) macurl2path 模块用于 URL 和 Macintosh 文件名的相互映射. 一般没有必要直接使用它, 请使用 urllib 中的机制. 它的用法参见 Example 13-14 .

13.14.0.1. Example 13-14. 使用 macurl2path 模块

```
File: macurl2path-example-1.py



import macurl2path
```

```

```

```
file = ":my:little:pony"



print macurl2path.pathname2url(file)

print
macurl2path.url2pathname(macurl2path.pathname2url
(file))



my/little/pony

:my:little:pony
```

13.15. nturl2path 模块

(功能实现模块) nturl2path 模块用于 URL 和 Windows 文件名的相互映射. 用法参见 Example 13-15 .

13.15.0.1. Example 13-15. 使用 nturl2path 模块

```
File: nturl2path-example-1.py



import nturl2path
```

```
file = r"c:\my\little\pony"
```

```


print nturl2path.pathname2url(file)

print
nturl2path.url2pathname(nturl2path.pathname2url(f
ile))



///C|/my/little/pony

C:\my\little\pony
```

同样地, 请通过 urllib 模块来访问这些函数, 如 Example 13-16 所示.

13.15.0.2. Example 13-16. 通过 urllib 调用 nturl2path 模块

```
File: nturl2path-example-2.py



import urllib



file = r"c:\my\little\pony"



print urllib.pathname2url(file)
```

```
print
urllib.url2pathname(urllib.pathname2url(file))
```

```
///C|/my/little/pony

C:\my\little\pony
```

13.16. tokenize 模块

tokenize 模块将一段 Python 源文件分割成不同的 token . 你可以在代码高亮工具中使用它.

在 Example 13-17 中, 我们分别打印出这些 token .

13.16.0.1. Example 13-17. 使用 tokenize 模块

```
File: tokenize-example-1.py



import tokenize



file = open("tokenize-example-1.py")



def handle_token(type, token, (srow, scol), (erow,
ecol), line):
```

```
    print "%d,%d-%d,%d:\t%s\t%s" % \
```

```
        (srow, scol, erow, ecol,
tokenize.tok_name[type], repr(token))



tokenize.tokenize(

    file.readline,

    handle_token

    )



1,0-1,6:     NAME    'import'
```

```
1,7-1,15:    NAME    'tokenize'
1,15-1,16:   NEWLINE '\012'
2,0-2,1:     NL      '\012'
3,0-3,4:     NAME    'file'
3,5-3,6:     OP      '='
3,7-3,11:    NAME    'open'
3,11-3,12:   OP      '('
3,12-3,35:   STRING  '"tokenize-example-1.py"'
3,35-3,36:   OP      ')'
3,36-3,37:   NEWLINE '\012'
...
```

注意这里的 tokenize 函数接受两个可调用对象作为参数: 前一个用于获取新的代码行, 第二个用于在获得每个 token 时调用.

13.17. keyword 模块

keyword 模块(参见 Example 13-18)有一个包含当前 Python 版本所使用的关键字的列表. 它还提供了一个字典, 以关键字作为 key , 以一个描述性函数作为 value , 它可用于检查给定单词是否是 Python 关键字.

13.17.0.1. Example 13-18. 使用 keyword 模块

File: keyword-example-1.py

```
import keyword

name = raw_input("Enter module name: ")

if keyword.iskeyword(name):
    print name, "is a reserved word."
    print "here's a complete list of reserved
words:"
    print keyword.kwlist
```

```
Enter module name: assert

assert is a reserved word.

here's a complete list of reserved words:

['and', 'assert', 'break', 'class', 'continue',
'def', 'del',

'elif','else','except','exec','finally','for',
'from',
```

```
'global', 'if', 'import', 'in', 'is', 'lambda',
'not', 'or',

'pass','print','raise','return','try','while']
```

13.18. parser 模块

(可选) parser 模块提供了一个到 Python 内建语法分析器和编译器的接口.

Example 13-19 将一个简单的表达式编译为一个抽象语法树(abstract syntax tree , AST), 然后将 AST 转换为一个嵌套列表, 转储树 (其中每个节点包含一个语法符号或者是一个 token)中的内容, 将所有数字加上 1 , 最后将列表转回一个代码对象. 至少我认为它是这么做的.

13.18.0.1. Example 13-19. 使用 parser 模块

```
File: parser-example-1.py



import parser

import symbol, token



def dump_and_modify(node):

    name = symbol.sym_name.get(node[0])

    if name is None:

        name = token.tok_name.get(node[0])
```

```
    print name,
    for i in range(1, len(node)):
        item = node[i]
        if type(item) is type([]):
            dump_and_modify(item)
        else:
            print repr(item)
            if name == "NUMBER":
                # increment all numbers!
                node[i] = repr(int(item)+1)

ast = parser.expr("1 + 3")

list = ast.tolist()

dump_and_modify(list)
```

```
ast = parser.sequence2ast(list)



print eval(parser.compileast(ast))
```

```
eval_input testlist test and_test not_test
comparison

expr xor_expr and_expr shift_expr arith_expr term
factor

power atom NUMBER '1'

PLUS '+'
```

```
term factor power atom NUMBER '3'
```

```
NEWLINE ''

ENDMARKER ''

6
```

13.19. symbol 模块

symbol 模块包含 Python 语法中的非终止符号. 可能只有你涉及 parser 模块的时候用到它. 用法参见 Example 13-20 .

13.19.0.1. Example 13-20. 使用 symbol 模块

```
File: symbol-example-1.py



import symbol



print "print", symbol.print_stmt

print "return", symbol.return_stmt



print 268
```

```
return 274
```

13.20. token 模块

token 模块包含标准 Python tokenizer 所使用的 token 标记. 如 Example 13-21 所示.

13.20.0.1. Example 13-21. 使用 token 模块

```
File: token-example-1.py



import token
```

```
print "NUMBER", token.NUMBER

print "PLUS", token.STAR

print "STRING", token.STRING



NUMBER 2

PLUS 16

STRING 3
```

14. 其他模块

14.1. 概览

本章描述的是一些并不怎么常见的模块. 一些是很实用的, 另些是已经废弃的模块.

14.2. pyclbr 模块

pyclbr 模块包含一个基本的 Python 类解析器, 如 Example 14-1 所示. 版本 1.5.2 中, 改模块只包含一个 readmodule 函数, 解析给定模块, 返回一个模块所有顶层类组成的列表.

14.2.0.1. Example 14-1. 使用 pyclbr 模块

```
File: pyclbr-example-1.py



import pyclbr



mod = pyclbr.readmodule("cgi")



for k, v in mod.items():
```

```
    print k, v
```

```
MiniFieldStorage <pyclbr.Class instance at 7873b0>

InterpFormContentDict <pyclbr.Class instance at
79bd00>

FieldStorage <pyclbr.Class instance at 790e20>

SvFormContentDict <pyclbr.Class instance at 79b5e0>

StringIO <pyclbr.Class instance at 77dd90>

FormContent <pyclbr.Class instance at 79bd60>

FormContentDict <pyclbr.Class instance at 79a9c0>
```

2.0 及以后版本中，添加了另个接口 readmodule_ex ，它还会读取全局函数. 如 Example 14-2 所示.

14.2.0.2. Example 14-2. 使用 pyclbr 模块读取类和函数

```
File: pyclbr-example-3.py



import pyclbr



# 2.0 and later
```

```
mod = pyclbr.readmodule_ex("cgi")
```

```


for k, v in mod.items():

    print k, v



MiniFieldStorage <pyclbr.Class instance at
00905D2C>

parse_header <pyclbr.Function instance at 00905BD4>

test <pyclbr.Function instance at 00906FBC>

print_environ_usage <pyclbr.Function instance at
00907C94>
```

```
parse_multipart <pyclbr.Function instance at
00905294>

FormContentDict <pyclbr.Class instance at 008D3494>

initlog <pyclbr.Function instance at 00904AAC>

parse <pyclbr.Function instance at 00904EFC>

StringIO <pyclbr.Class instance at 00903EAC>

SvFormContentDict <pyclbr.Class instance at
00906824>

...
```

访问类实例的属性可以获得关于类的更多信息，如 Example 14-3 所示.

14.2.0.3. Example 14-3. 使用 pyclbr 模块

```
File: pyclbr-example-2.py



import pyclbr

import string



mod = pyclbr.readmodule("cgi")
```

```
def dump(c):

    # print class header

    s = "class " + c.name

    if c.super:

        s = s + "(" + string.join(map(lambda v:
v.name, c.super), ", ") + ")"

    print s + ":"

    # print method names, sorted by line number

    methods = c.methods.items()

    methods.sort(lambda a, b: cmp(a[1], b[1]))

    for method, lineno in methods:

        print "  def " + method

    print



for k, v in mod.items():

    dump(v)
```

```
class MiniFieldStorage:

  def _ _init_ _

  def _ _repr_ _



class InterpFormContentDict(SvFormContentDict):

  def _ _getitem_ _

  def values

  def items



...
```

14.3. filecmp 模块

(2.0 新增) filecmp 模块用于比较文件和目录, 如 Example 14-4 所示.

14.3.0.1. Example 14-4. 使用 filecmp 模块

```
File: filecmp-example-1.py



import filecmp
```

```
if filecmp.cmp("samples/sample.au",
"samples/sample.wav"):

    print "files are identical"

else:

    print "files differ!"



# files differ!
```

1.5.2 以及先前版本中，你可以使用 cmp 和 dircmp 模块代替.

14.4. cmd 模块

cmd 模块为命令行接口(command-line interfaces , CLI)提供了一个简单的框架. 它被用在 pdb 模块中，当然你也可以在自己的程序中使用它，如 Example 14-5 所示.

你只需要继承 *Cmd* 类，定义 do 和 help 方法. 基类会自动地将这些方法转换为对应命令.

14.4.0.1. Example 14-5. 使用 cmd 模块

```
File: cmd-example-1.py



import cmd
```

```
import string, sys



class CLI(cmd.Cmd):



    def _ _init_ _(self):

        cmd.Cmd._ _init_ _(self)

        self.prompt = '> '
```

```
    def do_hello(self, arg):
```

```
        print "hello again", arg, "!"



    def help_hello(self):

        print "syntax: hello [message]",

        print "-- prints a hello message"



    def do_quit(self, arg):

        sys.exit(1)
```

```
    def help_quit(self):
        print "syntax: quit",
        print "-- terminates the application"

    # shortcuts
    do_q = do_quit
```

```
#
```

```
# try it out

cli = CLI()
cli.cmdloop()
```

```
> help

Documented commands (type help <topic>):
```

```
==========================================

hello           quit



Undocumented commands:

======================

help            q



> hello world
```

```
hello again world !
```

```
> q
```

14.5. rexec 模块

Feather 注: 版本 2.3 时取消了改模块的支持, 具体原因请参阅 : http://www.amk.ca/python/howto/rexec/ 和 http://mail.python.org/pipermail/python-dev/2002-December/031160.html

解决方法请参阅: http://mail.python.org/pipermail/python-list/2003-November/234581.html

`rexec` 模块提供了在限制环境下的 `exec`, `eval`, 以及 `import` 语句, 如 Example 14-6 所示. 在这个环境下, 所有可能对机器造成威胁的函数都不可用.

14.5.0.1. Example 14-6. 使用 rexec 模块

```
File: rexec-example-1.py



import rexec



r = rexec.RExec()

print r.r_eval("1+2+3")

print r.r_eval("__import__('os').remove('file')")
```

```

```

```
6

Traceback (innermost last):

  File "rexec-example-1.py", line 5, in ?

    print r.r_eval("__import__('os').remove('file')")

  File "/usr/local/lib/python1.5/rexec.py", line 257, in r_eval

    return eval(code, m.__dict__)

  File "<string>", line 0, in ?
```

```
AttributeError: remove
```

14.6. Bastion 模块

Feather 注: 版本 2.3 时取消了改模块的支持, 具体原因请参阅 : http://www.amk.ca/python/howto/rexec/ 和 http://mail.python.org/pipermail/python-dev/2003-January/031848.html

Bastion 模块, 允许你控制给定对象如何使用, 如 Example 14-7 所示. 你可以通过它把对象从未限制部分传递到限制部分.

默认情况下, 所有的实例变量都是隐藏的, 所有的方法以下划线开头.

14.6.0.1. Example 14-7. 使用 Bastion 模块

```
File: bastion-example-1.py
```

```

```

```
import Bastion



class Sample:

    value = 0



    def _set(self, value):

        self.value = value
```

```
    def setvalue(self, value):
        if 10 < value <= 20:
            self._set(value)
        else:
            raise ValueError, "illegal value"

    def getvalue(self):
        return self.value
```

```
#
# try it

s = Sample()
s._set(100) # cheat
print s.getvalue()
```

```
s = Bastion.Bastion(Sample())

s._set(100) # attempt to cheat

print s.getvalue()



100

Traceback (innermost last):

...

AttributeError: _set
```

你可以控制发布哪个函数. 在 Example 14- 中, 内部方法可以从外部调用, 但 getvalue 不再起作用.

14.6.0.2. Example 14-8. 使用 Bastion 模块处理非标准过滤器

```
File: bastion-example-2.py



import Bastion



class Sample:

    value = 0
```

```
    def _set(self, value):
        self.value = value

    def setvalue(self, value):
        if 10 < value <= 20:
            self._set(value)
        else:
            raise ValueError, "illegal value"
```

```
    def getvalue(self):
```

```
        return self.value


#
# try it


def is_public(name):
    return name[:3] != "get"
```

```
s = Bastion.Bastion(Sample(), is_public)

s._set(100) # this works

print s.getvalue() # but not this
```

```
100

Traceback (innermost last):

...

AttributeError: getvalue
```

14.7. readline 模块

(可选) readline 模块使用 GNU readline 库(或兼容库)实现了 Unix 下增强的输入编辑支持. 如 Example 14-9 所示.

该模块提供了增强的命令行编辑功能, 例如命令行历史等. 它还增强了 input 和 raw_input 函数.

14.7.0.1. Example 14-9. 使用 readline 模块

```
File: readline-example-1.py

import readline # activate readline editing
```

14.8. rlcompleter 模块

(可选, 只用于 Unix) rlcompleter 模块为 readline 模块提供了单词自动完成功能.

导入该模块就可以启动自动完成功能. 默认情况下完成函数被绑定在了 Esc 键上. 按两次 Esc 键就可以自动完成当前单词. 你可以使用下面的代码修改所绑定的键:

```
import readline

readline.parse_and_bind("tab: complete")
```

Example 14-10 展示了如何在程序中使用自动完成函数.

14.8.0.1. Example 14-10. 使用 rlcompleter 模块展开名字

```
File: rlcompleter-example-1.py



import rlcompleter

import sys



completer = rlcompleter.Completer()



for phrase in "co", "sys.p", "is":
```

```
    print phrase, "=>",
    # emulate readline completion handler
    try:
        for index in xrange(sys.maxint):
            term = completer.complete(phrase,
index)
            if term is None:
                break
            print term,
    except:
        pass
    print
```

```
co => continue compile complex coerce completer
sys.p => sys.path sys.platform sys.prefix
is => is isinstance issubclass
```

14.9. statvfs 模块

statvfs 模块包含一些与 os.statvfs (可选)函数配合使用的常量和函数, 该函数会返回文件系统的相关信息. 如 Example 14-11 所示.

14.9.0.1. Example 14-11. 使用 statvfs 模块

```
File: statvfs-example-1.py



import statvfs

import os
```

```
st = os.statvfs(".")
```

```


print "preferred block size", "=>",
st[statvfs.F_BSIZE]

print "fundamental block size", "=>",
st[statvfs.F_FRSIZE]

print "total blocks", "=>", st[statvfs.F_BLOCKS]

print "total free blocks", "=>",
st[statvfs.F_BFREE]

print "available blocks", "=>",
st[statvfs.F_BAVAIL]
```

```
print "total file nodes", "=>", st[statvfs.F_FILES]

print "total free nodes", "=>", st[statvfs.F_FFREE]

print "available nodes", "=>", st[statvfs.F_FAVAIL]

print "max file name length", "=>",
st[statvfs.F_NAMEMAX]
```

```
preferred block size => 8192

fundamental block size => 1024

total blocks => 749443
```

```
total free blocks => 110442
```

```
available blocks => 35497

total file nodes => 92158

total free nodes => 68164

available nodes => 68164

max file name length => 255
```

14.10. calendar 模块

`calendar` 模块是 Unix *cal* 命令的 Python 实现. 它可以将给定年份/月份的日历输出到标准输出设备上.

`prmonth(year, month)` 打印给定月份的日历, 如 Example 14-12 所示.

14.10.0.1. Example 14-12. 使用 calendar 模块

```
File: calendar-example-1.py



import calendar

calendar.prmonth(1999, 12)
```

```
    December 1999
```

```
Mo Tu We Th Fr Sa Su

       1  2  3  4  5

 6  7  8  9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30 31
```

`prcal(year)` 打印给定年份的日历, 如 Example 14-13 所示.

14.10.0.2. Example 14-13. 使用 calendar 模块

```
File: calendar-example-2.py



import calendar

calendar.prcal(2000)
```

```
                                  2000

       January                   February                  March

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su

                1  2            1  2  3  4  5  6         1  2  3  4  5

 3  4  5  6  7  8  9       7  8  9 10 11 12 13       6  7  8  9 10 11 12

10 11 12 13 14 15 16      14 15 16 17 18 19 20      13 14 15 16 17 18 19

17 18 19 20 21 22 23      21 22 23 24 25 26 27      20 21 22 23 24 25 26

24 25 26 27 28 29 30      28 29                     27 28 29 30 31
```

```
31



        April                     May                       June

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                             1  2       1  2  3  4  5  6  7      1  2  3  4
 3  4  5  6  7  8  9       8  9 10 11 12 13 14      5  6  7  8  9 10 11

10 11 12 13 14 15 16      15 16 17 18 19 20 21      12 13 14 15 16 17 18

17 18 19 20 21 22 23      22 23 24 25 26 27 28      19 20 21 22 23 24 25

24 25 26 27 28 29 30      29 30 31                  26 27 28 29 30



        July                      August                    September

Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                             1  2          1  2  3  4  5  6      1  2  3
```

```
 3  4  5  6  7  8  9        7  8  9 10 11 12 13
4  5  6  7  8  9 10

10 11 12 13 14 15 16       14 15 16 17 18 19 20       11
12 13 14 15 16 17

17 18 19 20 21 22 23       21 22 23 24 25 26 27       18
19 20 21 22 23 24

24 25 26 27 28 29 30       28 29 30 31
25 26 27 28 29 30

31
```

```
       October                    November
December
```

```
Mo Tu We Th Fr Sa Su       Mo Tu We Th Fr Sa Su       Mo
Tu We Th Fr Sa Su

                   1             1  2  3  4  5
1  2  3

 2  3  4  5  6  7  8        6  7  8  9 10 11 12
4  5  6  7  8  9 10

 9 10 11 12 13 14 15       13 14 15 16 17 18 19       11
12 13 14 15 16 17

16 17 18 19 20 21 22       20 21 22 23 24 25 26       18
19 20 21 22 23 24
```

```
23 24 25 26 27 28 29      27 28 29 30
25 26 27 28 29 30 31


30 31
```

注意这里的日历是按照欧洲习惯打印的，也就是说星期一是一个星期的第一天，其他情况需要请参考模块中的几个类.（和咱们一样，不用管了）

该模块中的其他类或函数可以帮助你输出自己需要的格式.

14.11. sched 模块

sched 模块为非线程环境提供了一个简单的计划任务模式. 如 Example 14-14 所示.

14.11.0.1. Example 14-14. 使用 sched 模块

```
File: sched-example-1.py
```

```

```

```
import sched

import time, sys



scheduler = sched.scheduler(time.time, time.sleep)



# add a few operations to the queue
```

```
scheduler.enter(0.5, 100, sys.stdout.write,
("one\n",))

scheduler.enter(1.0, 300, sys.stdout.write,
("three\n",))

scheduler.enter(1.0, 200, sys.stdout.write,
("two\n",))




scheduler.run()
```

```
one

two

three
```

14.12. statcache 模块

statcache 模块提供了访问文件相关信息的相关函数. 它是 os.stat 的扩展模块, 而且它会缓存收集到的信息. 如 Example 14-15 所示.

2.2 后该模块被废弃, 请使用 os.stat() 函数代替, 原因很简单, 它导致了更复杂的缓存管理, 反而降低了性能.

14.12.0.1. Example 14-15. 使用 statcache 模块

```
File: statcache-example-1.py
```

```
import statcache

import os, stat, time



now = time.time()

for i in range(1000):

    st = os.stat("samples/sample.txt")

print "os.stat", "=>", time.time() - now



now = time.time()

for i in range(1000):

    st = statcache.stat("samples/sample.txt")

print "statcache.stat", "=>", time.time() - now



print "mode", "=>",
oct(stat.S_IMODE(st[stat.ST_MODE]))

print "size", "=>", st[stat.ST_SIZE]

print "last modified", "=>",
time.ctime(st[stat.ST_MTIME])
```

```
os.stat => 0.371000051498

statcache.stat => 0.0199999809265

mode => 0666

size => 305

last modified => Sun Oct 10 18:39:37 1999
```

14.13. grep 模块

grep 模块提供了在文本文件中搜索字符串的另种方法, 如 Example 14-16 所示.

版本 2.1 时被声明不支持, 及就是说, 当前版本已经无法使用该模块.

14.13.0.1. Example 14-16. 使用 grep 模块

```
File: grep-example-1.py



import grep

import glob



grep.grep("\<rather\>",
glob.glob("samples/*.txt"))
```

```
# 4: indentation, rather than delimiters, might
become
```

14.14. dircache 模块

(已经废弃) 与 statcache 类似, 该模块是 os.listdir 函数的一个扩展, 提供了缓存支持, 可能因为同样的原因被废弃吧~ MUHAHAHAHAHA~~~~ . 请使用 os.listdir 代替. 如 Example 14-17 所示.

14.14.0.1. Example 14-17. 使用 dircache 模块

```
File: dircache-example-1.py
```

```
import dircache
```

```
import os, time

#
# test cached version
```

```
t0 = time.clock()



for i in range(100):

    dircache.listdir(os.sep)



print "cached", time.clock() - t0



#

# test standard version
```

```
t0 = time.clock()
```

```


for i in range(100):

    os.listdir(os.sep)



print "standard", time.clock() - t0


```

```
cached 0.0664509964968

standard 0.5560845807
```

14.15. dircmp 模块

(已废弃, 只用于 1.5.2) dircmp 模块用于比较两个目录的内容, 如 Example 14-18 所示.

14.15.0.1. Example 14-18. 使用 dircmp 模块

```
File: dircmp-example-1.py



import dircmp
```

```

```

```
d = dircmp.dircmp()

d.new("samples", "oldsamples")

d.run()

d.report()



diff samples oldsamples
```

```
Only in samples : ['sample.aiff', 'sample.au',
'sample.wav']


Identical files : ['sample.gif', 'sample.gz',
'sample.jpg', ...]
```

Python 2.0 后, 该模块被 filecmp 替换.

14.16. cmp 模块

(已废弃, 只用于 1.5.2) cmp 模块用于比较两个文件, 如 Example 14-19 所示.

14.16.0.1. Example 14-19. 使用 cmp 模块

```
File: cmp-example-1.py
```

```
import cmp
```

```


if cmp.cmp("samples/sample.au",
"samples/sample.wav"):

    print "files are identical"

else:

    print "files differ!"
```

```
files differ!
```

Python 2.0 后, 该模块被 filecmp 替换.

14.17. cmpcache 模块

(已废弃, 只用于 1.5.2) cmpcache 模块用于比较两个文件. 它是 cmp 模块的扩展, 提供了缓存支持. 如 Example 14-20 所示.

14.17.0.1. Example 14-20. 使用 cmpcache 模块

```
File: cmpcache-example-1.py



import cmpcache
```

```

```

```
if cmpcache.cmp("samples/sample.au",
"samples/sample.wav"):

    print "files are identical"

else:

    print "files differ!"
```

```
files differ!
```

Python 2.0 后, 该模块被 filecmp 替换.

但 filecmp 已经不提供缓存支持.

14.18. util 模块

(已废弃, 只用于 1.5.2) util 模块提供了常见操作的封装函数. 新代码可以使用如 Examples 14-21 到 14-23 的实现方法.

Example 14-21 展示了 remove(sequence, item) 函数.

14.18.0.1. Example 14-21. 实现 util 模块的 remove 函数

```
File: util-example-1.py
```

```
def remove(sequence, item):
```

```
    if item in sequence:

        sequence.remove(item)
```

Example 14-22 展示了 readfile(filename) => string 函数.

14.18.0.2. Example 14-22. 实现 util 模块的 readfile 函数

```
File: util-example-2.py
```

```
def readfile(filename):

    file = open(filename, "r")

    return file.read()
```

Example 14-23 展示了 `readopenfile(file) => string 函数.

14.18.0.3. Example 14-23. 实现 util 模块的 readopenfile 函数

```
File: util-example-3.py



def readopenfile(file):

    return file.read()
```

14.19. soundex 模块

(已废弃, 只用于 1.5.2) soundex 实现了一个简单的 hash 算法, 基于英文发音将单词转换为 6 个字符的字符串.

版本 2.0 后, 该模块已从标准库中删除.

get_soundex(word) 返回给定单词的 soundex 字符串.
sound_similar(word1, word2) 判断两个单词的 soundex 是否相同. 一般说来发音相似的单词有相同的 soundex . 如 Example 14-24 所示.

14.19.0.1. Example 14-24. 使用 soundex 模块

```
File: soundex-example-1.py


```

```
import soundex



a = "fredrik"

b = "friedrich"



print soundex.get_soundex(a),
soundex.get_soundex(b)



print soundex.sound_similar(a, b)
```

```

```

```
F63620 F63620

1
```

14.20. timing 模块

(已废弃, 只用于 Unix) timing 用于监控 Python 程序的执行时间. 如 Example 14-25 所示.

14.20.0.1. Example 14-25. 使用 timing 模块

```
File: timing-example-1.py
```

```
import timing
import time


def procedure():
    time.sleep(1.234)


timing.start()
```

```
procedure()
```

```
timing.finish()


print "seconds:", timing.seconds()
print "milliseconds:", timing.milli()
print "microseconds:", timing.micro()


seconds: 1
milliseconds: 1239
```

```
microseconds: 1239999
```

你可以按照 Example 14-26 中的方法用 time 模块实现 timing 模块的功能.

14.20.0.2. Example 14-26. 模拟 timing 模块

```
File: timing-example-2.py



import time




t0 = t1 = 0


def start():
    global t0
    t0 = time.time()




def finish():
    global t1
```

```
    t1 = time.time()



def seconds():

    return int(t1 - t0)



def milli():

    return int((t1 - t0) * 1000)



def micro():

    return int((t1 - t0) * 1000000)
```

time.clock() 可以替换 time.time() 获得 CPU 时间.

14.21. posixfile 模块

(已废弃, 只用于 Unix) posixfile 提供了一个类文件的对象(file-like object), 实现了文件锁定的支持. 如 Example 14-27 所示. 新程序请使用 fcntl 模块代替.

14.21.0.1. Example 14-27. 使用 posixfile 模块

```
File: posixfile-example-1.py
```

```
import posixfile
import string

filename = "counter.txt"

try:
    # open for update
    file = posixfile.open(filename, "r+")
    counter = int(file.read(6)) + 1
except IOError:
    # create it
    file = posixfile.open(filename, "w")
    counter = 0

file.lock("w|", 6)
```

```
file.seek(0) # rewind

file.write("%06d" % counter)



file.close() # releases lock
```

14.22. bisect 模块

bisect 模块用于向排序后的序列插入对象.

insort(sequence, item) 将条目插入到序列中, 并且保证序列的排序. 序列可以是任意实现了 __getitem__ 和 insert 方法的序列对象. 如 Example 14-28 所示.

14.22.0.1. Example 14-28. 使用 bisect 模块向列表插入条目

```
File: bisect-example-1.py



import bisect



list = [10, 20, 30]



bisect.insort(list, 25)

bisect.insort(list, 15)
```

```
print list
```

```
[10, 15, 20, 25, 30]
```

`bisect(sequence, item) => index` 返回条目插入后的索引值, 不对序列做任何修改. 如 Example 14-29 所示.

14.22.0.2. Example 14-29. 使用 bisect 模块获得插入点位置

```
File: bisect-example-2.py

import bisect

list = [10, 20, 30]

print list

print bisect.bisect(list, 25)

print bisect.bisect(list, 15)
```

```
[10, 20, 30]
```

```
2

1
```

14.23. knee 模块

knee 模块用于 Python 1.5 中导入包(package import)的实现. 当然 Python 解释器已经支持了这个, 所以这个模块几乎没有什么作用, 不过你可以看看它的代码, 明白这一切是怎么完成的.

代码请参见 Python-X.tgz\Python-2.4.4\Demo\imputil\knee.py

当然, 你可以导入该模块,如 Example 14-30 所示.

14.23.0.1. Example 14-30. 使用 knee 模块

```
File: knee-example-1.py



import knee



# that's all, folks!
```

14.24. tzparse 模块

(已废弃) tzparse 模块用于解析时区标志(time zone specification). 导入时它会自动分析 TZ 环境变量. 如 Example 14-31 所示.

14.24.0.1. Example 14-31. 使用 tzparse 模块

```
File: tzparse-example-1.py



import os
```

```
if not os.environ.has_key("TZ"):
```

```
    # set it to something...

    os.environ["TZ"] = "EST+5EDT;100/2,300/2"



# importing this module will parse the TZ variable

import tzparse



print "tzparams", "=>", tzparse.tzparams

print "timezone", "=>", tzparse.timezone

print "altzone", "=>", tzparse.altzone

print "daylight", "=>", tzparse.daylight

print "tzname", "=>", tzparse.tzname



tzparams => ('EST', 5, 'EDT', 100, 2, 300, 2)
```

```
timezone => 18000

altzone => 14400

daylight => 1

tzname => ('EST', 'EDT')
```

除了这些变量之外，该模块还提供了一些用于时间计算的函数.

14.25. regex 模块

(已废弃) `regex` 模块是旧版本的(1.5 前)正则表达式模块，用法如 Example 14-32 所示. 新代码请使用 `re` 模块实现.

注意在 Python 1.5.2 中 `regex` 比 `re` 模块要快. 但在新版本中 `re` 模块更快.

14.25.0.1. Example 14-32. 使用 regex 模块

```
File: regex-example-1.py



import regex



text = "Man's crisis of identity in the latter half
of the 20th century"
```

```
p = regex.compile("latter") # literal

print p.match(text)

print p.search(text), repr(p.group(0))
```

```

```

```
p = regex.compile("[0-9]+") # number

print p.search(text), repr(p.group(0))



p = regex.compile("\<\w\w\>") # two-letter word

print p.search(text), repr(p.group(0))



p = regex.compile("\w+$") # word at the end

print p.search(text), repr(p.group(0))



-1

32 'latter'

51 '20'

13 'of'
```

```
56 'century'
```

14.26. regsub 模块

(已废弃) `regsub` 模块提供了基于正则表达式的字符串替换操作. 用法如 Example 14-33 所示. 新代码请使用 `re` 模块中的 `replace` 函数代替.

14.26.0.1. Example 14-33. 使用 regsub 模块

```
File: regsub-example-1.py



import regsub



text = "Well, there's spam, egg, sausage, and spam."



print regsub.sub("spam", "ham", text) # just the
first

print regsub.gsub("spam", "bacon", text) # all of
them



Well, there's ham, egg, sausage, and spam.

Well, there's bacon, egg, sausage, and bacon.
```

14.27. reconvert 模块

(已废弃) reconvert 提供了旧样式正则表达式(regex 模块中使用)到新样式(re 模块)的转换工具. 如 Example 14-34 所示. 它也可以作为一个命令行工具.

14.27.0.1. Example 14-34. 使用 reconvert 模块

```
File: reconvert-example-1.py



import reconvert



for pattern in "abcd", "a\(b*c\)d", "\<\w+\>":

    print pattern, "=>",
reconvert.convert(pattern)



abcd => abcd

a\(b*c\)d => a(b*c)d

\<\w+\> => \b\w+\b
```

14.28. regex_syntax 模块

(已废弃) regex_syntax 模块用于改变正则表达式的模式, 如 Example 14-35 所示.

14.28.0.1. Example 14-35. 使用 regex_syntax 模块

```
File: regex-syntax-example-1.py



import regex_syntax

import regex



def compile(pattern, syntax):

    syntax = regex.set_syntax(syntax)

    try:

        pattern = regex.compile(pattern)

    finally:

        # restore original syntax

        regex.set_syntax(syntax)

    return pattern



def compile_awk(pattern):
```

```
    return compile(pattern,
regex_syntax.RE_SYNTAX_AWK)
```

```

```

```
def compile_grep(pattern):

    return compile(pattern,
regex_syntax.RE_SYNTAX_GREP)



def compile_emacs(pattern):

    return compile(pattern,
regex_syntax.RE_SYNTAX_EMACS)
```

14.29. find 模块

(已废弃, 只用于 1.5.2) find 模块用于在给定目录及其子目录中查找符合给定匹配模式的文件, 如 Example 14-36 所示.

匹配模式的语法与 fnmatch 中相同.

14.29.0.1. Example 14-36. 使用 find 模块

```
File: find-example-1.py



import find
```

```
# find all JPEG files in or beneath the current
directory
```

```
for file in find.find("*.jpg", ".":

    print file
```

```
.\samples\sample.jpg
```

15. Py 2.0 后新增模块

本章将在以后的时间里慢慢完成, 更新.

16. 后记