Python3 标准库概览

操作系统接口

os 模块提供了不少与操作系统相关联的函数。

```
>>> import os>>> os.getcwd()      # 返回当前的工作目录'C:\\Python34'>>> os.chdir('/server/accesslogs')   # 修改当前的工作目录>>> os.system('mkdir today')   # 执行系统命令 mkdir 0
```

建议使用 "import os" 风格而非 "from os import *"。这样可以保证随操作系统不同而有所变化的 os.open() 不会覆盖内置函数 open()。

在使用 os 这样的大型模块时内置的 dir() 和 help() 函数非常有用:

```
>>> import os>>> dir(os)<returns a list of all module functions>>>> help(os)<returns an extensive manual page created from the module's docstrings>
```

针对日常的文件和目录管理任务，:mod:shutil 模块提供了一个易于使用的高级接口:

```
>>> import shutil>>> shutil.copyfile('data.db', 'archive.db')>>> shutil.move('/build/executables', 'installdir')
```

文件通配符

glob 模块提供了一个函数用于从目录通配符搜索中生成文件列表:

```
>>> import glob>>> glob.glob('*.py')['primes.py', 'random.py', 'quote.py']
```

命令行参数

通用工具脚本经常调用命令行参数。这些命令行参数以链表形式存储于 sys 模块的 argv 变量。例如在命令行中执行 "python demo.py one two three" 后可以得到以下输出结果:

```
>>> import sys>>> print(sys.argv)['demo.py', 'one', 'two', 'three']
```

错误输出重定向和程序终止

sys 还有 stdin，stdout 和 stderr 属性，即使在 stdout 被重定向时，后者也可以用于显示警告和错误信息。

```
>>> sys.stderr.write('Warning, log file not found starting a new one\n')Warning, log file not found starting a new one
```

大多脚本的定向终止都使用 "sys.exit()"。

字符串正则匹配

re 模块为高级字符串处理提供了正则表达式工具。对于复杂的匹配和处理，正则表达式提供了简洁、优化的解决方案:

```
>>> import re>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')['foot', 'fell', 'fastest
']>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')'cat in the hat'
```

如果只需要简单的功能，应该首先考虑字符串方法，因为它们非常简单，易于阅读和调试:

```
>>> 'tea for too'.replace('too', 'two')'tea for two'
```

数学

math 模块为浮点运算提供了对底层 C 函数库的访问:

```
>>> import math>>> math.cos(math.pi / 4)0.70710678118654757>>> math.log(1024, 2)10.0
```

random 提供了生成随机数的工具。

```
>>> import random>>> random.choice(['apple', 'pear', 'banana'])'apple'>>> random.sample(range(100), 10)
 # sampling without replacement[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]>>> random.random()    # random floa
t0.17970987693706186>>> random.randrange(6)    # random integer chosen from range(6)4
```

访问 互联网

有几个模块用于访问互联网以及处理网络通信协议。其中最简单的两个是用于处理从 urls 接收的数据的 urllib.request 以及用于发送电子邮件的 smtplib:

```
>>> from urllib.request import urlopen>>> for line in urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.p
l'):...     line = line.decode('utf-8')  # Decoding the binary data to text....     if 'EST' in line or '
EDT' in line:  # look for Eastern Time...         print(line)

<BR>Nov. 25, 09:43:32 PM EST

>>> import smtplib>>> server = smtplib.SMTP('localhost')>>> server.sendmail('soothsayer@example.org', 'jc
aesar@example.org',... """To: jcaesar@example.org

... From: soothsayer@example.org

...

... Beware the Ides of March.

... """)>>> server.quit()
```

注意第二个例子需要本地有一个在运行的邮件服务器。

日期和时间

datetime 模块为日期和时间处理同时提供了简单和复杂的方法。

支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。

该模块还支持时区处理:

```
>>> # dates are easily constructed and formatted>>> from datetime import date>>> now = date.today()>>> no
w

datetime.date(2003, 12, 2)>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")'12-02-03. 0
2 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic>>> birthday = date(1964, 7, 31)>>> age = now - birthday>>> age.da
ys14368
```

数据压缩

以下模块直接支持通用的数据打包和压缩格式：zlib，gzip，bz2，zipfile，以及 tarfile。

```
>>> import zlib>>> s = b'witch which has which witches wrist watch'>>> len(s)41>>> t = zlib.compress
(s)>>> len(t)37>>> zlib.decompress(t)

b'witch which has which witches wrist watch'>>> zlib.crc32(s)226805979
```

性能度量

有些用户对了解解决同一问题的不同方法之间的性能差异很感兴趣。Python 提供了一个度量工具，为这些问题提供了直接答案。

例如，使用元组封装和拆封来交换元素看起来要比使用传统的方法要诱人的多,timeit 证明了现代的方法更快一些。

```
>>> from timeit import Timer>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()0.57535828626024577>>> Timer('
a,b = b,a', 'a=1; b=2').timeit()0.54962537085770791
```

相对于 timeit 的细粒度，:mod:profile 和 pstats 模块提供了针对更大代码块的时间度量工具。

测试模块

开发高质量软件的方法之一是为每一个函数开发测试代码，并且在开发过程中经常进行测试

doctest 模块提供了一个工具，扫描模块并根据程序中内嵌的文档字符串执行测试。

测试构造如同简单的将它的输出结果剪切并粘贴到文档字符串中。

通过用户提供的例子，它强化了文档，允许 doctest 模块确认代码的结果是否与文档一致:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)
import doctest doctest.testmod()   #
自动验证嵌入测试
```

unittest 模块不像 doctest 模块那么容易使用，不过它可以在一个独立的文件里提供一个更全面的测试集:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        self.assertRaises(ZeroDivisionError, average, [])
        self.assertRaises(TypeError, average, 20, 30, 70)

unittest.main() # Calling from the command line invokes all tests
```

Python3 面向对象

Python3 实例

1 篇笔记　写笔记

关于 **urlopen** 的补充

.

```
#处理 get 请求，不传 data，则为 get 请求 import urllibfrom urllib.request import
urlopenfrom urllib.parse import urlencode
```

```
url='http://www.xxx.com/login'
data={"username":"admin","password":123456}
req_data=urlencode(data)#将字典类型的请求数据转变为 url 编
码res=urlopen(url+'?'+req_data)#通过 urlopen 方法访问拼接好的 url
res=res.read().decode()#read()方法是读取返回数据内容，decode 是转换返回数据的 bytes 格式为
str print(res)#处理 post 请求,如果传了 data，则为 post 请求 import urllibfrom
urllib.request import Requestfrom urllib.parse import urlencode

url='http://www.xxx.com/login'
data={"username":"admin","password":123456} data=urlencode(data)#将
字典类型的请求数据转变为 url 编码data=data.encode('ascii')#将 url 编码
类型的请求数据转变为 bytes 类型
req_data=Request(url,data)#将 url 和请求数据处理为一个 Request 对象，供 urlopen 调用 with
urlope n(req_data) as res:
    res=res.read().decode()#read()方法是读取返回数据内容，decode 是转换返回数据的 bytes 格式为
str
print(res)
```

.