python 模块介绍- time 时间访问和转换

目录

项目简介

Python 中文库 https://bitbucket.org/xurongzhong/python-chinese-library 主要基于个人的使用经验，收集一些重要的外部和内部模块的中文教程和实例。发起人是 ouyangchongwu#gmail.com，xurongzhong#gmail.com 。

欢迎大家加入分享经验。联系方法：xurongzhong#gmail.com，微博：http://weibo.com/cizhenshi，python 及测试开发 qq 群 1：113938272，群 2:6089740。

文件下载：

1， https://bitbucket.org/xurongzhong/python-chinese-library/downloads。 推荐

2， hg clone 克隆所有文件 hg clone https://bitbucket.org/xurongzhong/python-chinese-library。

3， https://bitbucket.org/xurongzhong/python-chinese-library/src 浏览文件，右键点击文件，选另存为下载。

Bug 提交：https://bitbucket.org/xurongzhong/python-chinese-library/issuest。

版本管理

版本	修订发布时	修订人	备注

号	间		
V1.0	2013-12-04	Ouyangchongwu#gmail.com	初始版本，由《The Python Standard Library by Example 2011》和 http://docs.python.org/2.7/library/time.html 生成。

参考资料：

官方网址： http://docs.python.org/2.7/library/time.html

简介：

功能：时间访问和转换。

月下载量：

Python 版本：*Python 1.4 以上。*

当前版本：

下载地址：

平台：跨平台

相关模块：

datetime 标准模块。

calendar 标准模块。

Python-dateutil 日下载量 2 万左右，强烈推荐。

该模块提供的各种时间相关的函数。相关模块有 datetime 和 calendar。注意不是所有函数都是跨平台的。多数函数调用 c 库函数中的同名函数，所以参阅平台文档会有帮助。

下面介绍一些术语和约定：

epoch 是时间开始点。对于 Unix ，时代是 1970 年 1 月 1 日 0 点。通过 time.gmtime(0) 可以查看时间的起点：

```
In [2]: time.gmtime(0)
```

```
Out[2]: time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3,
tm_yday=1, tm_isdst=0)
```

对于 32 位的 linux 系统，时间只能处理到 2038 年。

```
>>> time.gmtime(time.time() + 786041553)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: timestamp out of range for platform time_t
```

UTC 是协调世界时（前身为格林威治标准时间或 GMT）。

DST 为夏令时，通常是根据当地法律在一年内的部分时间进行一小时的调整。 C 库包含有当地规则的表。

实时函数的精度可能比建议的要低。例如在大多数 Unix 系统中，时钟“滴答”只有 50 或 100 次每秒。

不过 time()和 sleep()比 Unix 的更好：时间为浮点数，time()的返回确保最精确(尽量使用 Unix 的函数 gettimeofday()) ，sleep()接受的时间为非零分数（尽量用 select()实现）。

gmtime(), localtime()和 strptime()的返回是包含 9 个整数的序列，可以作为 asctime(), mktime() and strftime()的输入，每个域都有自己的属性，实际上是一个结构体 struct_time，参见上面的例子。

时间转换：gmtime()把浮点时间转为 UTC 的 struct_time，反之 calendar.timegm()；localtime()把浮点时间转为 local 的 struct_time，反之 mktime()。实际上 calendar.timegm()和 mktime()是等效的，不过前者返回整数，后者返回浮点数。

时间生成与转换

生成 epoch 的浮点数：

```
In [30]: time.time()
Out[30]: 1386057314.581948
```

注意不同的系统精度不同，linux 一般是小数点后面 7 为，windows 一般是小数点后 3 位。Time 函数是没有参数的。可以直接对返回的浮点数进行计算：

```
In [31]: time.gmtime(time.time() + 786041553)
```

```
Out[31]: time.struct_time(tm_year=2038, tm_mon=10, tm_mday=31, tm_hour=0, tm_min=57, tm_sec=18,
tm_wday=6, tm_yday=304, tm_isdst=0)
```

上述命令因为处理年限的关系，在 32 位 linux 上面会报错：

```
>>> time.gmtime(time.time() + 786041553)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: timestamp out of range for platform time_t
```

上述命令中的 gmtime([secs])把浮点时间转为 UTC 的 struct_time，如果无输入参数为空会调用 time()读取当前时间，如：

```
In [32]: time.gmtime()
Out[32]: time.struct_time(tm_year=2013, tm_mon=12, tm_mday=3, tm_hour=8, tm_min=8, tm_sec=46,
tm_wday=1, tm_yday=337, tm_isdst=0)
```

上面显示的是世界协调时间， localtime([secs])可以显示本地时间：

```
In [34]: time.localtime()
Out[34]: time.struct_time(tm_year=2013, tm_mon=12, tm_mday=3, tm_hour=16, tm_min=14, tm_sec=40,
tm_wday=1, tm_yday=337, tm_isdst=0)
```

注意夏时制要设置 dst。asctime([t])显示时间为可读性好的格式，它 gmtime(), localtime()和 strptime()的返回的 struct_time 类型转换为可读性较好的格式。如果输入参数为空则调用 localtime()的返回结果。它和 c 函数不同的地方是末尾不会添加换行。asctime 不会使用 Locale 信息。

```
In [35]: time.asctime()
Out[35]: 'Tue Dec  3 16:18:02 2013'
```

ctime([secs])在 asctime 上更进一步，转换浮点数为可读性较好的格式，相当于 asctime(localtime(secs)), 这个功能很常用。ctime 不会使用 Locale 信息。

```
In [33]: time.ctime()
Out[33]: 'Tue Dec  3 16:12:37 2013'
```

Sleep

sleep(secs)暂停执行指定秒数。参数可以是整数或浮点数。实际的中止时间可能小

于请求时间，因为例行的信号捕捉可能终止 sleep。此外中止时间可能长于请求时间，因为因为系统调度也是需要时间的。

```
In [36]: time.sleep(3)
```

处理器时间

clock()在 Unix 上，返回当前的处理器时间，为以秒表示的浮点数。精度决于同名的 C 函数，通常用于基准 Python 或定时的算法。我们书写一个不耗 cpu 和耗 cpu 的脚本对比：

```
import time

for i in range(6, 1, -1):
    print '%s %0.2f %0.2f' % (time.ctime(),
                              time.time(),
                              time.clock())
    print 'Sleeping', i
    time.sleep(i)
```

执行结果：

```
# python time_clock_sleep.py

Tue Dec  3 16:34:55 2013 1386059695.94 0.07

Sleeping 6

Tue Dec  3 16:35:01 2013 1386059701.94 0.07

Sleeping 5

Tue Dec  3 16:35:06 2013 1386059706.95 0.07

Sleeping 4

Tue Dec  3 16:35:10 2013 1386059710.95 0.07

Sleeping 3

Tue Dec  3 16:35:13 2013 1386059713.95 0.07

Sleeping 2
```

```
import hashlib
import time

# Data to use to calculate md5 checksums
data = open(__file__, 'rt').read()

for i in range(5):
    h = hashlib.sha1()
    print time.ctime(), ': %0.3f %0.3f' % (time.time(), time.clock())
    for i in range(300000):
        h.update(data)
    cksum = h.digest()
```

执行结果：

```
]# python time_clock.py

Tue Dec  3 16:27:24 2013 : 1386059244.364 0.090

Tue Dec  3 16:27:25 2013 : 1386059245.187 0.910

Tue Dec  3 16:27:25 2013 : 1386059245.943 1.660

Tue Dec  3 16:27:26 2013 : 1386059246.698 2.420

Tue Dec  3 16:27:27 2013 : 1386059247.452 3.170
```

struct_time 类

struct_time 是的命名元组，结构如下：

索引（Index）	属性（Attribute）	值（Values）
0	tm_year（年）	比如 2013
1	tm_mon（月）	1 - 12
2	tm_mday（日）	1 - 31

3	tm_hour（时）	0 - 23
4	tm_min（分）	0 - 59
5	tm_sec（秒）	0 - 61
6	tm_wday（weekday）	0 - 6（0 表示周日）
7	tm_yday（一年中的第几天）	1 - 366
8	tm_isdst（是否是夏令时）	默认为-1

```
import time

def show_struct(s):
    print '  tm_year :', s.tm_year
    print '  tm_mon  :', s.tm_mon
    print '  tm_mday :', s.tm_mday
    print '  tm_hour :', s.tm_hour
    print '  tm_min  :', s.tm_min
    print '  tm_sec  :', s.tm_sec
    print '  tm_wday :', s.tm_wday
    print '  tm_yday :', s.tm_yday
    print '  tm_isdst:', s.tm_isdst

print 'gmtime:'
show_struct(time.gmtime())
print '\nlocaltime:'
show_struct(time.localtime())
print '\nmktime:', time.mktime(time.localtime())
```

执行结果：

```
# python time_struct.py

gmtime:

  tm_year : 2013
```

```
  tm_mon   : 12
  tm_mday : 3
  tm_hour : 9
  tm_min   : 55
  tm_sec   : 54
  tm_wday : 1
  tm_yday : 337
  tm_isdst: 0

localtime:
  tm_year : 2013
  tm_mon   : 12
  tm_mday : 3
  tm_hour : 17
  tm_min   : 55
  tm_sec   : 54
  tm_wday : 1
  tm_yday : 337
  tm_isdst: 0

mktime: 1386064554.0
```

时区

重置库函数的时间转换规则。实际上是修改环境变量 TZ，python 2.3 以后类 linux 支持该功能，这个功能相对不是那么常用。TZ 环境变量的格式如下：

```
std offset [dst [offset [,start[/time], end[/time]]]]
```

STD 和 DST 为时区缩写。hh[:mm[:ss]]，表示加上这个时间可以得到 UTC 时间。偏移量的形式为： HH [： MM [： SS]]，夏时制增加 1 小时。start[/time]，end[/time]表示使用夏时制的区间。time 和偏移类似，默认时间是 02:00:00。比如：

```
>>> os.environ['TZ'] = 'EST+05EDT,M4.1.0,M10.5.0'

>>> time.tzset()

>>> time.strftime('%X %x %Z')

'02:07:36 05/08/03 EDT'

>>> os.environ['TZ'] = 'AEST-10AEDT-11,M10.5.0,M3.5.0'

>>> time.tzset()

>>> time.strftime('%X %x %Z')

'16:08:12 05/08/03 AEST'
```

在许多 Unix 系统（包括* BSD，Linux 和 Solaris，和 Darwin），使用系统时区数据库更方便。

```
>>> os.environ['TZ'] = 'US/Eastern'

>>> time.tzset()

>>> time.tzname

('EST', 'EDT')

>>> os.environ['TZ'] = 'Egypt'

>>> time.tzset()

>>> time.tzname

('EET', 'EEST')
```

另一实例：

```
import time

import os

def show_zone_info():
    print '  TZ    :', os.environ.get('TZ', '(not set)')
    print '  tzname:', time.tzname
```

```
    print '  Zone  : %d (%d)' % (time.timezone,
                                 (time.timezone / 3600))
    print '  DST   :', time.daylight
    print '  Time  :', time.ctime()
    print

print 'Default :'
show_zone_info()

ZONES = [ 'GMT',
          'Europe/Amsterdam',
          ]

for zone in ZONES:
    os.environ['TZ'] = zone
    time.tzset()
    print zone, ':'
    show_zone_info()
```

执行结果：

```
# python time_timezone.py
Default:
  TZ      : (not set)
  tzname: ('CST', 'CST')
  Zone    : -28800 (-8)
  DST     : 0
  Time    : Wed Dec  4 11:13:26 2013


GMT:
  TZ      : GMT
  tzname: ('GMT', 'GMT')
  Zone    : 0 (0)
  DST     : 0
```

```
    Time    : Wed Dec    4 03:13:26 2013


Europe/Amsterdam:

    TZ      : Europe/Amsterdam

    tzname: ('CET', 'CEST')

    Zone    : -3600 (-1)

    DST     : 1

    Time    : Wed Dec    4 04:13:26 2013
```

格式化

time.strftime(format[, t])：把一个代表时间的元组或者 struct_tim 转为格式化的时间字符串。如果 t 未指定，将调用 time.localtime()的返回作为输入。如果输入中任何一个元素越界将报 ValueError 异常。格式化参数如下：

格式	含义	备注
%a	本地简化星期名	
%A	本地完整星期名	
%b	本地简化月份名	
%B	本地完整月份名称	
%c	本地相应的日期和时间表示	
%d	日期（01 - 31）	

%H	小时（24 小时制，00-23）	
%I	小时（12 小时制，01-12）	
%j	天数（基于年）（001-366）	
%m	月份（01-12）	
%M	分钟（00-59）	
%p	显示 am 或 pm 的标识	一
%S	秒（01-61）	二
%U	周数（基于年）（00–53 周日是星期的开始。）第一个周日之前的所有天数都放在第 0 周。	三
%w	星期中的天数（0-6，0 是星期天）	三
%W	和%U 基本相同，以星期一为星期的开始。	
%x	本地相应日期表示	
%X	本地相应时间表示	
%y	去掉世纪的年份（00-99）	
%Y	完整的年份	

%Z	时区的名字（如果不存在为空字符）	
%%	‘%’字符	

备注：

“%p”只有与“%I”配合使用才有效果。

秒是 0 - 61，而不是 59，以处理闰秒和双闰秒。

当使用 strptime()函数时，只有当在这年中的周数和天数被确定的时候%U 和%W 才会被计算。

比如：

```
In [12]: time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())

Out[12]: 'Wed, 04 Dec 2013 03:34:58 +0000'
```

显示格式可能因系统而又不同的差异。

time.strptime(string[, format])：把一个格式化时间字符串转化为 struct_time。实际上它和 strftime()是逆操作，参数参见 strftime。Format 默认为"%a %b %d %H:%M:%S %Y"，和 ctime 的返回格式一致，没有提供的值会采用默认值(1900, 1, 1, 0, 0, 0, 0, 1, -1)。

```
In [14]: time.strptime("30 Nov 00", "%d %b %y")

Out[14]: time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30, tm_hour=0, tm_min=0, tm_sec=0,
tm_wday=3, tm_yday=335, tm_isdst=-1)
```

其他

altzone 属性查看当前夏时制时间的偏移。daylight 属性查看是否使用了夏时制。timezone 查看当前时区的偏移。Tzname 返回本地时区和夏时制对应的时区。

```
In [3]: time.altzone

Out[3]: -28800

In [4]: time.daylight

Out[4]: 0

In [5]: time.timezone
```

```
Out[5]: -28800

In [6]: time.tzname

Out[6]: ('CST', 'CST')
```