学习Python Doc第十天: 标准库巡礼(二)

目录

之前对标准库中的部分模块已经有了简单的介绍，见这里 。今天，我们浏览剩下的部分。我只快速浏览，详细的用法留到工程实践中去。所以这部分内容大概可以五分钟过完。

1 输出格式

reprlib 模块提供了一个 repr() 的版本，用于显示较大的容器的简写。

```
>>> import reprlib
>>> reprlib.repr(set('supercalifragilisticexpialidocious'))
"{'a', 'c', 'd', 'e', 'f', 'g', ...}"
```

在输出内置的或者用户定义的对象时， pprint 模块提供了更精巧的控制。当结果冲过一行时， pretty printer 会自动断行，并美化显示结果。

textwrap 模块会格式化一段文字，使其适合在给定的屏幕上显示。

2 模板

string 模块提供了 Template 类。这个类的语法简单，适合普通用户使用。格式字符创中使用 $ 定位 Pytone 变量位置。看代码：

```
>>> from string import Template
>>> t = Template('${village}folk send $$10 to $cause.')
>>> t.substitute(village='Nottingham', cause='the ditch fund')
'Nottinghamfolk send $10 to the ditch fund.'
```

当一个$ 后的变量没有被替换时， substitude() 方法会产生一个 KeyError 的错误。

3 二进制文件工具

struct 模块提供了 pack() 和 unpack() 函数。这两个函数可以用来操作二进制文件。

```
import struct

with open('myfile.zip', 'rb') as f:
    data = f.read()

start = 0
for i in range(3):                      # show the first 3 file headers
    start += 14
    fields = struct.unpack('<IIIHH', data[start:start+16])
    crc32, comp_size, uncomp_size, filenamesize, extra_size = fields

    start += 16
    filename = data[start:start+filenamesize]
    start += filenamesize
    extra = data[start:start+extra_size]
    print(filename, hex(crc32), comp_size, uncomp_size)

    start += extra_size + comp_size     # skip to the next header
```

4 多线程

多线程使得并行计算成为可能。 threading 模块提供了很多函数用于产生多线程。

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile

    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print('Finished background zip of:', self.infile)

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print('The main program continues to run in foreground.')

background.join()    # Wait for the background task to finish
print('Main program waited until background was done.')
```

多线程编程的最大挑战是协调多个线程的数据和其他计算资源。 threading 模块提供了很多同步机制用来保证数据一致性，这些同步机制包括：locks,events,condition variables, semaphores

5 日志

logging 模块提供了全能且灵活的日志系统。最简单的情况是：用文件或者 sys.stderr 来记录日志。

```
import logging
```


```
logging.debug('Debugging information')
logging.info('Informational message')
logging.warning('Warning:config file %s not found', 'server.conf')
logging.error('Error occurred')
logging.critical('Critical error -- shutting down')
```

输出为：

```
WARNING:root:Warning:config file server.conf not found
ERROR:root:Error occurred
CRITICAL:root:Critical error -- shutting down
```

6 弱引用

Python 提供自动内存管理机制，有自己的 garbage collection 系统。对大多数对象执行 reference counting 。当最后一个reference消失的时候，对象的内存被释放。

这套机制在大多数应用中都工作良好。但是，偶尔情况下，我们需要对某个对象进行长时间的追踪。不幸的是，紧紧追踪这些对象会产生一个永远也不能消除reference。 weakref 模块提供了追踪对象而不生成reference的方法。当一个对象不再使用，它会从弱引用表中删除。