PyMOTW: StringIO and cStringIO

模块：StringIO 和 cStringIO Purpose: Work with text buffers using file-like API 目的：类似于file操作的文本缓冲区API Python版本：StringIO: 1.4, cStringIO: 1.5

描述：

The StringIO class provides a convenient means of working with text in-memory using the file API (read, write. etc.). There are 2 separate implementations. The cStringIO module is written in C for speed, while the StringIO module is written in Python for portability. Using cStringIO to build large strings can offer performance savings over some other string concatenation techniques. 类StringIO提供了一个在内存中方便处理文本的类文件(读, 写等操作)API. 他有两个独立的实现, 一个是用c实现的cStringIP模块,速度较快, 另一个是StringIO模块,他用python实现的以增强其可移植性. 使用cStringIO来处理大字符串可以提高运行性能,优于其他字符串串联技术.

例子： Here are some pretty standard, simple, examples of using StringIO buffers: 这里是一个好的,标准的,简单的.使用StringIO缓冲的例子:

#!/usr/bin/env python

"""Simple examples with StringIO module 使用StringIO模块简单的例子 """

Find the best implementation available on this platform找到此平台上最好的StringIO实现模块

try:

from cStringIO import StringIO

except: from StringIO import StringIO

Writing to a buffer写入一个缓冲区 output = StringIO() output.write('This goes into the buffer. ') ##sy: write方法将字符串写入缓冲区

print »»output, 'And so does this.' ##sy: print 语句将字符串写入output # Retrieve the value written查询刚才被写入缓冲区的值 print output.getvalue()

output.close() # discard buffer memory关闭(丢弃)缓冲区

Initialize a read buffer 初始化一个只读缓冲区 input = StringIO('Inital value for read buffer')

Read from the buffer 从缓冲区中读取数据 print input.read()

This example uses read(), but of course the readline() and readlines() methods are also available. The StringIO class also provides a seek() method so it is possible to jump around in a buffer while reading, which can be useful for rewinding if you are using some sort of look-ahead parsing algorithm. 这个例子中使用了read(),但当然,函数readline()和readlines()也都是可用的.类StringIO提供seek()函数,因此在读取数据时可以任意跳到某个点上, 这也当你使用某些前看解析算法可以回头读取.

Real world applications of StringIO include a web application stack where various parts of the stack may add text to the response, or testing the output generated by parts of a program which typically write to a file. 现实世界中,StringIO的应用包括一个网络应用程序栈,栈的各个部分都可以增加文本到响应response对象中,或者测试由程序某段输出(典型是写入到文件)的数据.

The application we are building at work includes a shell scripting interface in the form of several command line programs. Some of these programs are responsible for pulling data from the database and dumping it on the console (either to show the user, or so the text can serve as input to another command). The commands share a set of formatter plugins to produce a text representation of an object in a variety of ways (XML, bash syntax, human readable, etc.). 我们想在创建的工程应用中包括一个shell脚本接口, 他是以多个命令行程序的形式.这些程序中的某些是负责从数据库中取数据,然后将这些数据转储到控制台(可以是现实给用户,也可以是文本中,这样就可以作为另一个命令的输入). 这些命令他们共享了一组格式化插件以便产生一个对象的多种文本表示(XML, bash语法, 人可读的,等等). Since the formatters normally write to standard output, testing the results would be a little tricky without the StringIO module. Using StringIO to intercept the output of the formatter gives us an easy way to collect the output in memory to compare against expected results. 因为格式化可以将输出数据标准化后写到标准输出,如果没有StringIO模块,所得的测试结果可能会有些奇怪. 而使用了StringIO来拦截格式化的输出,这样以便我们用一个简单的方式来收集内存中的输出数据,来对预期的结果做比较.

参考:

* The StringIO module ::: www.effbot.org

* Efficient String Concatenation in Python
* Python Module of the Week

Updated 5/20/2007 with technorati tags. Updated 9/5/2007 with minor formatting changes.