**Python XML与Excel转换**

1. 解析XML

Python对XML的解析方法有多种，如SAX、DOM、ElementTree

1. ElementTree（元素树）解析xml

ElementTree就像一个轻量级的DOM，具有方便友好的API，代码可用性号，速度快，消耗内存少

1. 加载xml文件

加载xml文件共有2中方法

# 加载指定字符串

root = ElementTree.fromstring(text)

# 加载指定文件

root = ElementTree.parse(r”E:/Learning/Outcomes/PyTest/src/幻视APP测试用例-3.11.xml”)

1. 获取element

获取element的方法如下

# 通过getiterator

lst\_node = root.getiterator(“testcase”)

for node in lst\_node:

print\_node(node)

# 通过getchildren

lst\_node\_child = lst\_node[0].getchildren()[0]

print\_node(lst\_node\_child)

# find方法

node\_find = root.find(‘testcase’)

print\_node(node\_find)

# findall方法

node\_findall = root.findall(“testcase/summary”)[1]

print\_node(node\_findall)

if \_\_name\_\_ == ‘\_\_main\_\_’:

read\_xml(open(“E:/Learning/Outcomes/PyTest/src/幻视APP测试用例-3.11.xml”).read())

举例：

1. #encoding=utf-8
2. **from** xml.etree **import** ElementTree as ET
3. #要找出所有人的年龄
4. per=ET.parse('test.xml')
5. p=per.findall('/person')
6. **for** x **in** p:
7. **print** x.attrib
8. **print**
9. **for** oneper **in** p:  #找出person节点
10. **for** child **in** oneper.getchildren(): #找出person节点的子节点
11. **print** child.tag,':',child.text
13. **print** 'age:',oneper.get('age')
14. **print** '############'

结果

1. {'age': '18'}
2. {'age': '19', 'des': 'hello'}
4. name : hzj
5. sex : man
6. age: 18
7. ############
8. name : kiki
9. sex : female
10. age: 19
11. ############
12. SAX解析xml

python 标准库包含SAX解析器，SAX用事件驱动模型，通过在解析XML的过程中触发一个个的事件并调用用户定义的回调函数来处理XML文件。SAX流式读取XML文件，比较快，占用内存少，但需要用户实现回调函数（handler）。

利用SAX解析XML文档牵涉到两个部分:解析器和事件处理器。解析器负责读取XML文档,并向事件处理器发送事件,如元素开始跟元素结束事件;而事件处理器则负责对事件作出相应,对传递的XML数据进行处理。

适合处理以下问题：

①对大型文件进行处理

②只需要文件的部分内容，或只需从文件中得到特定信息

③想建立自己的对象模型的时候

在python中使用sax方式处理xml要先引入xml.sax中的parse函数，还有xml.sax.handler中的ContentHandler。

1. ContenHandler类
2. characters（content）方法

调用时机：

从行开始，遇到标签之前，存在字符，content的值为这些字符串

从一个标签，遇到下一个标签之前，存在字符，content的值为这些字符串

从一个标签，遇到行结束符之前，存在字符，content的值为这些字符串

1. startDocument（）方法：文档启动时调用
2. endDocument（）方法：解析器到达文档结尾时调用
3. startElement（name，attrs）方法：遇到xml开始标签时调用，name为标签名字，attrs是标签属性值字典
4. endElement（name）方法： 遇到xml结束 标签时调用
5. make\_parse方法

#创建一个新的解析器对象并返回，parsee\_list为可选参数，解析器列表

xml.sax.make\_parser（[parsee\_list]）

1. parser方法

#创建一个SAX解析器并解析xml文档

xml.sax.parser(xmlfile,contenthandler[,errorhandler])

* 参数说明

xmlfile –xml文件名

contenthandle –必须是一个ContentHandler的对象

errorhandler –若指定 该参数，则errorhandler必须是一个SAX ErrorHandler对象

1. parseString方法

#创建一个XML解析器并解析xml字符串

xml.sax.parseString(xmlstring, contenthandler[, errorhandler])

* 参数说明

xmlstring–xml字符串

contenthandle –必须是一个ContentHandler的对象

errorhandler –若指定 该参数，则errorhandler必须是一个SAX ErrorHandler对象

xml实例

<collection shelf="New Arrivals">

<movie title="Enemy Behind">

<type>War, Thriller</type>

<format>DVD</format>

<year>2003</year>

<rating>PG</rating>

<stars>10</stars>

<description>Talk about a US-Japan war</description>

</movie>

<movie title="Transformers">

<type>Anime, Science Fiction</type>

<format>DVD</format>

<year>1989</year>

<rating>R</rating>

<stars>8</stars>

<description>A schientific fiction</description>

</movie>

<movie title="Trigun">

<type>Anime, Action</type>

<format>DVD</format>

<episodes>4</episodes>

<rating>PG</rating>

<stars>10</stars>

<description>Vash the Stampede!</description>

</movie>

<movie title="Ishtar">

<type>Comedy</type>

<format>VHS</format>

<rating>PG</rating>

<stars>2</stars>

<description>Viewable boredom</description>

</movie>

</collection>

xml解析实例

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

import xml.sax

class MovieHandler( xml.sax.ContentHandler ):

def \_\_init\_\_(self):

self.CurrentData = ""

self.type = ""

self.format = ""

self.year = ""

self.rating = ""

self.stars = ""

self.description = ""

# 元素开始事件处理

def startElement(self, tag, attributes):

self.CurrentData = tag

if tag == "movie":

print "\*\*\*\*\*Movie\*\*\*\*\*"

title = attributes["title"]

print "Title:", title

# 元素结束事件处理

def endElement(self, tag):

if self.CurrentData == "type":

print "Type:", self.type

elif self.CurrentData == "format":

print "Format:", self.format

elif self.CurrentData == "year":

print "Year:", self.year

elif self.CurrentData == "rating":

print "Rating:", self.rating

elif self.CurrentData == "stars":

print "Stars:", self.stars

elif self.CurrentData == "description":

print "Description:", self.description

self.CurrentData = ""

# 内容事件处理

def characters(self, content):

if self.CurrentData == "type":

self.type = content

elif self.CurrentData == "format":

self.format = content

elif self.CurrentData == "year":

self.year = content

elif self.CurrentData == "rating":

self.rating = content

elif self.CurrentData == "stars":

self.stars = content

elif self.CurrentData == "description":

self.description = content

if ( \_\_name\_\_ == "\_\_main\_\_"):

# 创建一个 XMLReader

parser = xml.sax.make\_parser()

# turn off namepsaces

parser.setFeature(xml.sax.handler.feature\_namespaces, 0)

# 重写 ContextHandler

Handler = MovieHandler()

parser.setContentHandler( Handler )

parser.parse("movies.xml")

1. xml.dom解析xml

DOM (Document Object Model)文件对象模型，解析器在解析xml时，一次性读取整个文档，将XML数据在内存中解析成一个树，通过对树的操作来操作XML。因DOM需要将XML数据映射到内存中的树，一是比较慢，二是比较耗内存。

Python中用xml.dom.minidom来解析xml

# 打开xml文件，并将这个文件对象dom变量

dom = xml.dom.minidom.parsee(‘xxx.xml‘’’)

# 得到dom对象的文档元素，并把获得的对象给root

root = dom.documentElement

#对于知道元素名字的子元素，可以使用getElementsByTagName方法获取

bb = root.getElementsByTagName(‘标签子元素名’)

每一个结点都有它的nodeName，nodeValue，nodeType属性。nodeValue只对文本结点有效

# 对于xml中相同标签名字的标签区分

bb = root.getElementsByTagName(‘标签名1’)

b = bb[2]

print b.nodeName

bb = root.getElementsByTagName(‘标签名2’)

b = bb[1]

print b.nodeName

这里root.getElementsByTagName(‘标签名1’)获得的标签为标签1的一组标签，而b[0]表示一组标签中的第一个；b[2] ，表示这一组标签中的第三个。

#获得标签的属性

有些标签是有属性的，getAttribute()方法可以活着元素的属性对于的值

，如下：

un = item.getAttribute(“属性值1”)

#获得标签对之间的数据

xml标签，一般都是成对出现，其标签对之间的有有数据的，我们可以有多种方法获取标签对之间的数据，如下

①

#!/usr/bin/python

# -\*- coding: UTF-8 -\*-

from xml.dom.minidom import parse

import xml.dom.minidom

# 使用minidom解析器打开 XML 文档

DOMTree = xml.dom.minidom.parse("movies.xml")

collection = DOMTree.documentElement

if collection.hasAttribute("shelf"):

print "Root element : %s" % collection.getAttribute("shelf")

# 在集合中获取所有电影

movies = collection.getElementsByTagName("movie")

# 打印每部电影的详细信息

for movie in movies:

print "\*\*\*\*\*Movie\*\*\*\*\*"

if movie.hasAttribute("title"):

print "Title: %s" % movie.getAttribute("title")

type = movie.getElementsByTagName('type')[0]

print "Type: %s" % type.childNodes[0].data

format = movie.getElementsByTagName('format')[0]

print "Format: %s" % format.childNodes[0].data

rating = movie.getElementsByTagName('rating')[0]

print "Rating: %s" % rating.childNodes[0].data

description = movie.getElementsByTagName('description')[0]

print "Description: %s" % description.childNodes[0].data