XML IN PYTHON

Processing Xml Docs in Python

Mohammadreza Shaghouzi

Sh.mohammad66@gmail.com

Parsing VS. Processing

- Parsing : breaks down a text into recognized strings of characters for further analysis.

- Processing : operations that will allow you not just to parse, but to apply some kind of transformation to the text.

Which XML library to use?

- **xml.parsers.expat** - Fast XML parsing using Expat
- **xml.dom** - The Document Object Model API
- **xml.dom.minidom** - Lightweight DOM implementation
- **xml.dom.pulldom** - Support for building partial DOM trees
- **xml.sax** - Support for SAX2 parsers
- **xml.sax.handler** - Base classes for SAX handlers
- **xml.sax.saxutils** - SAX Utilities
- **xml.sax.xmlreader** - Interface for XML parsers
- **xml.etree.ElementTree** - The ElementTree XML API

ElementTree Functions

- xml.etree.ElementTree.**Comment**(text=None)

Comment element factory.

- xml.etree.ElementTree.**dump**(elem)

Writes an element tree or element structure to sys.stdout. This function should be used for debugging only. The exact output format is implementation dependent. In this version, it's written as an ordinary XML file.

elem is an element tree or an individual element.

- xml.etree.ElementTree.**fromstring**(text)

Parses an XML section from a string constant. Same as XML(). text is a string containing XML data. Returns an Element instance.

ElementTree Functions

- xml.etree.ElementTree.**fromstringlist**(sequence, parser=None)

Parses an XML document from a sequence of string fragments. sequence is a list or other sequence containing XML data fragments. parser is an optional parser instance. If not given, the standard XMLParser parser is used. Returns an Element instance.

- xml.etree.ElementTree.**iselement**(element)

Checks if an object appears to be a valid element object. element is an element instance. Returns a true value if this is an element object.

- xml.etree.ElementTree.**parse**(source, parser=None)

Parses an XML section into an element tree. source is a filename or file object containing XML data. parser is an optional parser instance. If not given, the standard XMLParser parser is used. Returns an ElementTree instance.

ElementTree Functions

- xml.etree.ElementTree.**SubElement**(parent, tag, attrib={}, **extra)

Subelement factory. This function creates an element instance with its atrributes, and appends it to an existing element. Returns an element instance.

- xml.etree.ElementTree.**tostring**(element, encoding="us-ascii", method="xml")

Generates a string representation of an XML element, including all subelements. element is an Element instance. encoding [1] is the output encoding (default is US-ASCII). method is either "xml", "html" or "text" (default is "xml"). Returns an encoded string containing the XML data.

- xml.etree.ElementTree.**tostringlist**(element, encoding="us-ascii", method="xml")

Generates a string representation of an XML element, including all subelements. Returns a list of encoded strings containing the XML data

Element Objects

- tag

A string identifying what kind of data this element represents (the element type, in other words).

- text
- tail

These attributes can be used to hold additional data associated with the element. Their values are usually strings but may be any application-specific object. If the element is created from an XML file, the text attribute holds either the text between the element's start tag and its first child or end tag, or None, and the tail attribute holds either the text between the element's end tag and the next tag, or None. For the XML data

- attrib

A dictionary containing the element's attributes.

Element Objects

- **get**(key, default=None)

Gets the element attribute named key.

Returns the attribute value, or default if the attribute was not found.

- **items**()

Returns the element attributes as a sequence of (name, value) pairs. The attributes are returned in an arbitrary order.

- **keys**()

Returns the elements attribute names as a list. The names are returned in an arbitrary order.

- **set**(key, value)

Set the attribute key on the element to value.

The following methods work on the element's children (subelements).

Element Objects

- **append**(subelement)

Adds the element subelement to the end of this elements internal list of subelements.

- **extend**(subelements)

Appends subelements from a sequence object with zero or more elements. Raises AssertionError if a subelement is not a valid object.

- **find**(match)

Finds the first subelement matching match. match may be a tag name or path. Returns an element instance or None.

- **findall**(match)

Finds all matching subelements, by tag name or path. Returns a list containing all matching elements in document order.

Element Objects

- **insert**(index, element)

Inserts a subelement at the given position in this element.

- **iter**(tag=None)

Creates a tree iterator with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order. If tag is not None or '*', only elements whose tag equals tag are returned from the iterator. If the tree structure is modified during iteration, the result is undefined.

- **remove**(subelement)

Removes subelement from the element. Unlike the find* methods this method compares elements based on the instance identity, not on tag value or contents.

ElementTree Objects

- **_setroot**(element)

Replaces the root element for this tree. This discards the current contents of the tree, and replaces it with the given element. Use with care.

- **find**(match)

Same as Element.find(), starting at the root of the tree.

- **findall**(match)
- **getroot**()

Returns the root element for this tree.

- **iter**(tag=None)

Creates and returns a tree iterator for the root element. The iterator loops over all elements in this tree, in section order. tag is the tag to look for (default is to return all elements).

ElementTree Objects

- **iterfind**(match)

Finds all matching subelements, by tag name or path. Same as getroot().iterfind(match). Returns an iterable yielding all matching elements in document order.

- **parse**(source, parser=None)

Loads an external XML section into this element tree. source is a file name or file object. parser is an optional parser instance. If not given, the standard XMLParser parser is used. Returns the section root element.

- **write**(file, encoding="us-ascii", xml_declaration=None, default_namespace=None, method="xml")

Writes the element tree to a file, as XML. file is a file name, or a file object opened for writing. encoding [1] same as tostring().

Using Methods

- Library: xml.etree.elementtree

Default in Python Core (no need to install)

- IDE: PyCharm(Python 2.7)

Also You could use idle python

- Sample xml file(test.xml)

Sample Xml(test.xml)

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Parsing XML

- Reading From Disk

```
import xml.etree.ElementTree as ET
tree = ET.parse('test.xml')
root = tree.getroot()
```

- Reading From String

```
root = ET.fromstring(test)
```

- Print Tag & Attribue

```
for child in root:
    print child.tag,child.attrib
```

```
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

- Access with specific index

```
print root[0][1].text
```

```
2008
```

Finding interesting elements

- Using Element.iter():

```
for item in root.iter('neighbor'):
    print item.attrib
```

```
{'direction': 'E', 'name': 'Austria'}
{'direction': 'W', 'name': 'Switzerland'}
{'direction': 'N', 'name': 'Malaysia'}
{'direction': 'W', 'name': 'Costa Rica'}
{'direction': 'E', 'name': 'Colombia'}
```

- Using Element.findall():

```
for item in root.findall('country'):
    rank = item.find('rank').text
    name = item.get('name')
    print name,rank
```

```
Liechtenstein 1
Singapore 4
Panama 68
```

Modifying an XML File

- Update Element

```
for rank in root.iter('rank'):
    new_rank=int(rank.text)+1
    rank.text=str(new_rank)
    rank.set('updated','yes')
tree.write('output.xml')
```

```
<?xml version="1.0"?>
<data>
   <country name="Liechtenstein">
      <rank updated="yes">2</rank>
      <year>2008</year>
      <gdppc>141100</gdppc>
      <neighbor name="Austria" direction="E"/>
      <neighbor name="Switzerland"
direction="W"/>
   </country>
   <country name="Singapore">
      <rank updated="yes">5</rank>
      <year>2011</year>
      <gdppc>59900</gdppc>
      <neighbor name="Malaysia" direction="N"/>
   </country>
 <country name="Panama">
      <rank updated="yes">69</rank>
      <year>2011</year>
      <gdppc>13600</gdppc>
      <neighbor name="Costa Rica"
direction="W"/>
      <neighbor name="Colombia" direction="E"/>
   </country>
</data>
```

Modifying an XML File

- Remove Element

```
for country in root.findall('country'):
    rank=int(country.find('rank').text)
    if rank >50:
        root.remove(country)
tree.write('output.xml')
```

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank updated="yes">2</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria"
direction="E"/>
        <neighbor name="Switzerland"
direction="W"/>
    </country>
    <country name="Singapore">
        <rank updated="yes">5</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia"
direction="N"/>
    </country>
</data>
```

Building XML documents

- Dump Element Tree

```
a = ET._Element('Information Retrieval')
b=ET.SubElement(a,'Cylinder?!')
b.text ="HEll Yeah"
c=ET.SubElement(a,'shiar')
c.text="No"
aa=ET._Element('Man')
toor=ET._Element('root')
toor.extend((a,aa))
ete =ET.dump(toor)
```

Output in console:

```
<root><Information Retrieval><Cylinder?!>HEll
Yeah</Cylinder?!><shiar>No</shiar></Information Retrieval><Man /></root>
```

Building XML documents

- Save to output file

```
a = ET._Element('Information Retrieval')
b=ET.SubElement(a,'Cylinder?!')
b.text ="HEll Yeah"
c=ET.SubElement(a,'shiar')
c.text="No"
aa=ET._Element('Man')
toor=ET._Element('root')
toor.extend((a,aa))
ete =ET.dump(toor)
```

Output Content:

```
<root><Information Retrieval><Cylinder?!>HEll
Yeah</Cylinder?!><shiar>No</shiar></Information Retrieval><Man /></root>
```

DataBase in Python

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000 and later
- Informix
- Interbase
- Oracle
- Sybase

Export DB To XML

- Connect and Fetch Records

Rely on which database use

- Write to the xml file

Its not Complex; just write them to file.

In this case, we are using SQL SERVER2012 and **pypyodbc** lib for sql connection.

Export DB To XML

- Fetch and writing to a file

```
import pypyodbc
connection = pypyodbc.connect('Driver={SQL Server};'
                              'Server=ASUS\MOHAMMADSH;'
                              'Database=Entekhabat;'
                              'uid=sa;pwd=P@ssw0rd')
#Fetch
cursor = connection.cursor()
sqlcmd= "SELECT * FROM IR"
cursor.execute(sqlcmd)
columns = [i[0] for i in cursor.description]
allRows = cursor.fetchall()
```

Export DB To XML

```
#Writing to file
xmlFile = open('backup.xml','w')
xmlFile.write('<?xml version="1.0" ?>\n')
xmlFile.write('<IR>')
for rows in allRows:
	xmlFile.write('<row>')
	columnNumber = 0
	for column in columns:
		data = rows[columnNumber]
		if data == None:
			data = ''
		xmlFile.write('<%s>%s</%s>' % (column,data,column))
		columnNumber += 1
	xmlFile.write('</row>')
xmlFile.write('</IR>')
xmlFile.close()
```

Export DB To XML

- Execute T-SQL command(in SQL Server)

```
import pypyodbc
connection = pypyodbc.connect('Driver={SQL Server};'
                              'Server=ASUS\MOHAMMADSH;'
                              'Database=Entekhabat;'
                              'uid=kenpachi;pwd=P@ssw0rd')
value=[]
cursor = connection.cursor()

with open('exportxml.sql','r') as file:
    var1 = file.read().strip().replace("\r\n","")
sqlcmd=var1
print sqlcmd
cursor.execute(sqlcmd)
cursor.commit()
file.close()
connection.close()
```

Export DB to XML

- Exportxml.sql

```
DECLARE @OutputFile NVARCHAR(100) ,    @FilePath NVARCHAR(100) ,
@bcpCommand NVARCHAR(1000)

SET @bcpCommand = 'bcp "SELECT * FROM Entekhabat.dbo.IR FOR XML
PATH" queryout '
SET @FilePath = 'G:\Projects\exmpopencv\'
SET @OutputFile = 'result.xml'
SET @bcpCommand = @bcpCommand + @FilePath + @OutputFile + ' -x -c
-t, -T -S '+ @@SERVERNAME
exec master..xp_cmdshell @bcpCommand
```

- Note: xp_cmdshell has default disabled.

Export DB To XML

- Result:

<row><irid>1</irid><irfname>Mohammadreza</irfname><irlname>shaghouzi</irlname><irgrade>1.800000000000000e+001</irgrade></row><row><irid>2</irid><irfname>Mohammad amin</irfname><irlname>bajand</irlname><irgrade>2.000000000000000e+001</irgrade></row><row><irid>3</irid><irfname>mohammad hosein</irfname><irlname>ghaznavi</irlname><irgrade>2.000000000000000e+001</irgrade></row>

Import DB from XML

- Parse Xml file and Insert Into the Table

```
import pypyodbc
connection = pypyodbc.connect('Driver={SQL Server};'
                              'Server=ASUS\MOHAMMADSH;'
                              'Database=Entekhabat;'
                              'uid=kenpachi;pwd=P@ssw0rd')
cursor = connection.cursor()
import xml.etree.ElementTree as ET
tree = ET.parse('backup.xml')
root = tree.getroot()
id=[]
fname=[]
lname=[]
grade=[]
```

Import DB from XML

```
#INSERTION
for item in root.findall('row'):
    id.append(item.find('irid').text)
    fname.append(item.find('irfname').text)
    lname.append(item.find('irlname').text)
    grade.append(item.find('irgrade').text)
for i in range(0,len(id)):
    sqlcmd="INSERT INTO IR(irid,irfname,irlname,irgrade)VALUES
(?,?,?,?)"
    values=[id[i],fname[i],lname[i],grade[i]]
    cursor.execute(sqlcmd,values)
connection.commit()
connection.close()
```

Import DB From XML

- Execute T-SQL Command

```
import pypyodbc
connection = pypyodbc.connect('Driver={SQL Server};'
                              'Server=ASUS\MOHAMMADSH;'
                              'Database=Entekhabat;'
                              'uid=kenpachi;pwd=P@ssw0rd')
value=[]
cursor = connection.cursor()
with open('importxml.sql','r') as file:
    var1 = file.read().strip().replace("\r\n","")
sqlcmd=var1
cursor.execute(sqlcmd)
cursor.commit()
file.close()
connection.close()
```

Import DB From XML

- Importxml.sql

```
DECLARE @messagebody XML
SELECT @messagebody = BulkColumn
FROM OPENROWSET(BULK 'G:\Projects\exmpopencv\result.xml',
SINGLE_CLOB) AS X

INSERT INTO [dbo].[IR]
select a.value(N'(./irid)[1]', N'int') as [IRid],
       a.value(N'(./irfname)[1]', N'nvarchar(50)') as [IRfname],
       a.value(N'(./irlname)[1]', N'nvarchar(50)') as [IRlname],
       a.value(N'(./irgrade)[1]', N'float') as [IRgrade]
from @messagebody.nodes('/row') as r(a)
```

Import DB From XML

- Result:

ASUS\MOHAMMADSH...khabat - dbo.IR | importxml.sql - A...US\Mohamn

	irid	irfname	irlname	irgrade
▶	1	Mohammad...	shaghouzi	18
	2	Mohammad...	bajand	20
	3	mohammad...	ghaznavi	20
*	*NULL*	*NULL*	*NULL*	*NULL*

Thanks...

Keep Calm and code Python :)