Python - Processamento XML

XML é uma linguagem portátil e de código aberto que permite aos programadores desenvolver aplicativos que podem ser lidos por outros aplicativos, independentemente do sistema operacional e/ou linguagem de desenvolvimento.

O que é XML?

A Extensible Markup Language (XML) é uma linguagem de marcação muito parecida com HTML ou SGML. Isto é recomendado pelo World Wide Web Consortium e está disponível como um padrão aberto.

XML é extremamente útil para controlar pequenas e médias quantidades de dados sem exigir um backbone baseado em SQL.

Arquiteturas e APIs do analisador XML.

A biblioteca padrão do Python fornece um conjunto mínimo, mas útil, de interfaces para trabalhar com XML. Todos os submódulos para processamento XML estão disponíveis no pacote xml.

- **xml.etree.ElementTree** - a API ElementTree, um processador XML simples e leve
- **xml.dom** - a definição da API DOM.
- **xml.dom.minidom** - uma implementação mínima de DOM.
- **xml.dom.pulldom** - suporte para construção de árvores DOM parciais.
- **xml.sax** - Classes básicas SAX2 e funções de conveniência.
- **xml.parsers.expat** - a ligação do analisador Expat.

As duas APIs mais básicas e amplamente utilizadas para dados XML são as interfaces SAX e DOM.

- **Simple API for XML (SAX)** - Aqui, você registra retornos de chamada para eventos de interesse e deixa o analisador prosseguir no documento. Isso é útil quando seus documentos são grandes ou você tem limitações de memória, ele analisa o arquivo à medida que o lê no disco e o arquivo inteiro nunca é armazenado na memória.
- **Document Object Model (DOM)** - Esta é uma recomendação do World Wide Web Consortium em que o arquivo inteiro é lido na memória e armazenado em um formato hierárquico (baseado em árvore) para representar todos os recursos de um documento XML.

Obviamente, o SAX não pode processar informações tão rápido quanto o DOM, ao trabalhar com arquivos grandes. Por outro lado, usar exclusivamente DOM pode realmente matar seus recursos, especialmente se usado em muitos arquivos pequenos.

SAX é somente leitura, enquanto DOM permite alterações no arquivo XML. Como essas duas APIs diferentes literalmente se complementam, não há razão para que você não possa usá-las em projetos grandes.

Para todos os nossos exemplos de código XML, vamos usar um arquivo XML simples **movies.xml** como entrada -

```
<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>
```

Analisando XML com APIs SAX

SAX é uma interface padrão para análise XML orientada a eventos. A análise de XML com SAX geralmente requer que você crie seu próprio ContentHandler subclassificando xml.sax.ContentHandler.

Seu ContentHandler lida com tags e atributos específicos de seus tipos de XML. Um objeto ContentHandler fornece métodos para lidar com vários eventos de análise. Seu analisador proprietário chama métodos ContentHandler enquanto analisa o arquivo XML.

Os métodos startDocument e endDocument são chamados no início e no final do arquivo XML. O método caracteres(texto) recebe os dados dos caracteres do arquivo XML por meio do parâmetro texto.

O ContentHandler é chamado no início e no final de cada elemento. Se o analisador não estiver no modo namespace, os métodos startElement(tag, atributos) eendElement(tag) serão chamados; caso contrário, os métodos correspondentesstartElementNS e endElementNS serão chamados. Aqui, tag é a tag do elemento e atributos é um objeto Atributos.

Aqui estão outros métodos importantes para entender antes de prosseguir -

O método make_parser

O método a seguir cria um novo objeto analisador e o retorna. O objeto analisador criado será do primeiro tipo de analisador, segundo o sistema.

```
xml.sax.make_parser( [parser_list] )
```

Aqui estão os detalhes dos parâmetros -

- **parser_list** - O argumento opcional que consiste em uma lista de analisadores a serem usados, que devem implementar o método make_parser.

O método de análise

O método a seguir cria um analisador SAX e o utiliza para analisar um documento.

```
xml.sax.parse( xmlfile, contenthandler[, errorhandler])
```

Aqui estão os detalhes dos parâmetros -

- **xmlfile** - Este é o nome do arquivo XML para leitura.
- **contenthandler** - Este deve ser um objeto ContentHandler.
- **errorhandler** - Se especificado, errorhandler deve ser um objeto SAX ErrorHandler.

O método parseString

Existe mais um método para criar um analisador SAX e analisar a string XML especificada.

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

Aqui estão os detalhes dos parâmetros -

- **xmlstring** - Este é o nome da string XML a ser lida.
- **contenthandler** - Este deve ser um objeto ContentHandler.
- **errorhandler** - Se especificado, errorhandler deve ser um objeto SAX ErrorHandler.

Exemplo

```
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 = ""

   # Call when an element starts
   def startElement(self, tag, attributes):
      self.CurrentData = tag
      if tag == "movie":
         print ("*****Movie*****")
         title = attributes["title"]
         print ("Title:", title)

   # Call when an elements ends
   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 = ""

   # Call when a character is read
   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__"):

   # create an XMLReader
   parser = xml.sax.make_parser()

   # turn off namepsaces
   parser.setFeature(xml.sax.handler.feature_namespaces, 0)

   # override the default ContextHandler
   Handler = MovieHandler()
   parser.setContentHandler( Handler )

   parser.parse("movies.xml")
```

Isso produziria o seguinte resultado -

```
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
```

```
Year: 1989
Rating: R
Stars: 8
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Stars: 10
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Stars: 2
Description: Viewable boredom
```

Para obter detalhes completos sobre a documentação da API SAX, consulte APIs SAX padrão do Python .

Analisando XML com APIs DOM

O Document Object Model ("DOM") é uma API de linguagem cruzada do World Wide Web Consortium (W3C) para acessar e modificar os documentos XML.

O DOM é extremamente útil para aplicações de acesso aleatório. SAX permite visualizar apenas um bit do documento por vez. Se você estiver olhando para um elemento SAX, não terá acesso a outro.

Esta é a maneira mais fácil de carregar um documento XML rapidamente e criar um objeto minidom usando o módulo xml.dom. O objeto minidom fornece um método de análise simples que cria rapidamente uma árvore DOM a partir do arquivo XML.

A frase de exemplo chama a função parse(file [,parser]) do objeto minidom para analisar o arquivo XML, designado por file em um objeto de árvore DOM.

```
from xml.dom.minidom import parse
import xml.dom.minidom

# Open XML document using minidom parser
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
if collection.hasAttribute("shelf"):
   print ("Root element : %s" % collection.getAttribute("shelf"))
```

```
# Get all the movies in the collection
movies = collection.getElementsByTagName("movie")

# Print detail of each movie.
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)
```

Isso produziria a seguinte **saída** -

```
Root element : New Arrivals
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Rating: PG
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A schientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Description: Vash the Stampede!
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Description: Viewable boredom
```

Para obter detalhes completos sobre a documentação da API DOM, consulte APIs DOM padrão do Python .

API XML ElementTree

O pacote xml possui um módulo ElementTree. Esta é uma API de processador XML simples e leve.

XML é um formato de dados hierárquicos semelhante a uma árvore. O 'ElementTree' neste módulo trata todo o documento XML como uma árvore. A classe 'Element' representa um único nó nesta árvore. As operações de leitura e gravação em arquivos XML são feitas no nível ElementTree. As interações com um único elemento XML e seus subelementos são feitas no nível do Elemento.

Crie um arquivo XML

A árvore é uma estrutura hierárquica de elementos começando pela raiz seguida por outros elementos. Cada elemento é criado usando a função Element() deste módulo.

```
import xml.etree.ElementTree as et
e=et.Element('name')
```

Cada elemento é caracterizado por uma tag e um atributo attrib que é um objeto dict. Para o elemento inicial da árvore, attrib é um dicionário vazio.

```
>>> root=xml.Element('employees')
>>> root.tag
'employees'
>>> root.attrib
{}
```

Agora você pode configurar um ou mais elementos filhos para serem adicionados ao elemento raiz. Cada filho pode ter um ou mais subelementos. Adicione-os usando a função SubElement() e defina seu atributo de texto.

```
child=xml.Element("employee")
nm = xml.SubElement(child, "name")
nm.text = student.get('name')
age = xml.SubElement(child, "salary")
age.text = str(student.get('salary'))
```

Cada filho é adicionado ao root pela função append() como -

```
root.append(child)
```

Depois de adicionar o número necessário de elementos filhos, construa um objeto de árvore pela função elementTree() -

```
tree = et.ElementTree(root)
```

Toda a estrutura da árvore é gravada em um arquivo binário pela função write() do objeto árvore -

```
f=open('employees.xml', "wb")
tree.write(f)
```

Exemplo

Neste exemplo, uma árvore é construída a partir de uma lista de itens de dicionário. Cada item do dicionário contém pares de valores-chave que descrevem uma estrutura de dados do aluno. A árvore assim construída é gravada em 'myfile.xml'

```
import xml.etree.ElementTree as et
employees=[{'name':'aaa','age':21,'sal':5000},{'name':xyz,'age':22,'sal':6000}]
root = et.Element("employees")
for employee in employees:
   child=xml.Element("employee")
   root.append(child)
   nm = xml.SubElement(child, "name")
   nm.text = student.get('name')
   age = xml.SubElement(child, "age")
   age.text = str(student.get('age'))
   sal=xml.SubElement(child, "sal")
   sal.text=str(student.get('sal'))
tree = et.ElementTree(root)
with open('employees.xml', "wb") as fh:
   tree.write(fh)
```

O 'myfile.xml' é armazenado no diretório de trabalho atual.

```
<employees><employee><name>aaa</name><age>21</age><sal>5000</sal></employee><employee
```


Analisar um arquivo XML

Vamos agora reler o 'myfile.xml' criado no exemplo acima. Para tanto, serão utilizadas as seguintes funções do módulo ElementTree -

ElementTree() - Esta função está sobrecarregada para ler a estrutura hierárquica dos elementos para objetos de uma árvore.

```
tree = et.ElementTree(file='students.xml')
```

getroot() - Esta função retorna o elemento raiz da árvore.

```
root = tree.getroot()
```

Você pode obter a lista de subelementos um nível abaixo de um elemento.

```
children = list(root)
```

No exemplo a seguir, os elementos e subelementos de 'myfile.xml' são analisados em uma lista de itens de dicionário.

Exemplo

```
import xml.etree.ElementTree as et
tree = et.ElementTree(file='employees.xml')
root = tree.getroot()
employees=[]
   children = list(root)
for child in children:
   employee={}
   pairs = list(child)
   for pair in pairs:
      employee[pair.tag]=pair.text
   employees.append(employee)
print (employees)
```

Ele produzirá a seguinte **saída** -

```
[{'name': 'aaa', 'age': '21', 'sal': '5000'}, {'name': 'xyz', 'age':'22', 'sal': '6000'}]
```


Modificar um arquivo XML

Usaremos a função iter() do Elemento. Ele cria um iterador de árvore para determinada tag com o elemento atual como raiz. O iterador itera sobre este elemento e todos os elementos abaixo dele, na ordem do documento (profundidade primeiro).

Vamos construir um iterador para todos os subelementos 'marcas' e incrementar o texto de cada tag sal em 100.

```
import xml.etree.ElementTree as et
tree = et.ElementTree(file='students.xml')
root = tree.getroot()
for x in root.iter('sal'):
   s=int (x.text)
   s=s+100
   x.text=str(s)
with open("employees.xml", "wb") as fh:
   tree.write(fh)
```

Nosso 'employees.xml' agora será modificado de acordo. Também podemos usar set() para atualizar o valor de uma determinada chave.

```
x.set(marks, str(mark))
```