IPC2
201903878

DOM XML PYTHON

El Modelo de Objetos del Documento, o «DOM» por sus siglas en inglés, es un lenguaje API del Consorcio *World Wide Web* (W3C) para acceder y modificar documentos XML. Una implementación del DOM presenta los documento XML como un árbol, o permite al código cliente construir dichas estructuras desde cero para luego darles acceso a la estructura a través de un conjunto de objetos que implementaron interfaces conocidas.

Las aplicaciones DOM típicamente empiezan al diseccionar (*parse*) el XML en un DOM. Cómo esto funciona no está incluido en el DOM nivel 1, y el nivel 2 provee mejoras limitadas. Existe una clase objeto llamada DOMImplementation que da acceso a métodos de creación de Document, pero de ninguna forma da acceso a los constructores (*builders*) de *reader*/*parser*/*Document* de una forma independiente a la implementación. No hay una forma clara para acceder a estos método sin un objeto Document existente.
En Python, cada implementación del DOM proporcionará una función getDOMImplementation(). El DOM de nivel 3 añade una especificación para Cargar(*Load*)/Guardar(*Store*), que define una interfaz al lector (*reader*), pero no está disponible aún en la librería estándar de Python.

Objetos en el DOM

La documentación definitiva para el DOM es la especificación del DOM del W3C.

Los atributos del DOM también pueden ser manipulados como nodos en vez de simples cadenas de caracteres (*strings*).

Interfaz	Sección	Propósito
`DOMImplementation`	Objetos DOMImplementation	Interfaz para las implementaciones subyacentes.
`Node`	Objetos Nodo	Interfaz base para la mayoría de objetos en un documento.
`NodeList`	Objetos NodeList	Interfaz para una secuencia de nodos.
`DocumentType`	Objetos DocumentType	Información acerca de la declaraciones necesarias

Interfaz	Sección	Propósito
		para procesar un documento.
`Document`	Objetos Documento	Objeto que representa un documento entero.
`Element`	Objetos Elemento	Nodos elemento en la jerarquía del documento.
`Attr`	Objetos Atributo	Nodos de los valores de los atributos en los elementos nodo.
`Comment`	Objetos Comentario	Representación de los comentarios en el documento fuente.
`Text`	Objetos Texto y CDATASection	Nodos con contenido textual del documento.
`ProcessingInstruction`	Objetos ProcessingInstruction	Representación de instrucción del procesamiento.

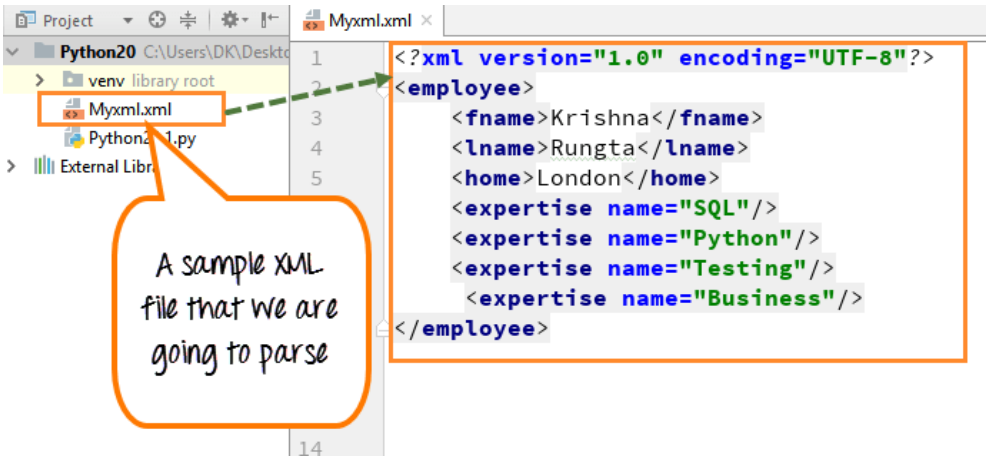
USO DE LA LIBRERÍA

Cómo analizar XML usando minidom

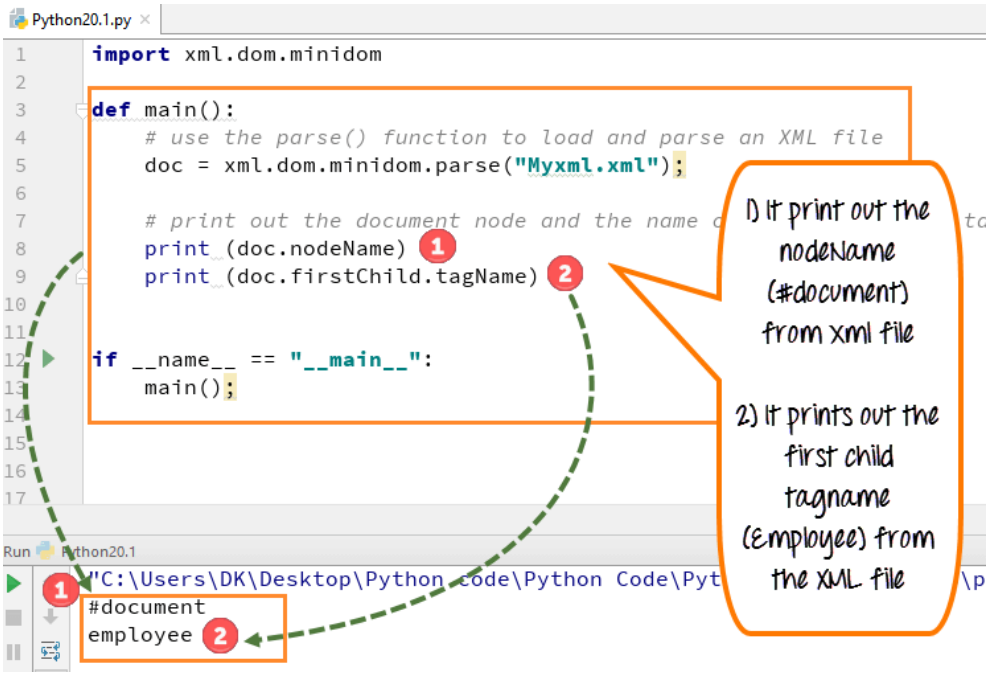
Hemos creado un archivo XML de muestra que vamos a analizar.

Paso 1) Dentro del archivo, podemos ver el nombre, apellido, casa y el área de especialización (SQL, Python, Testing y Business)

IPC2
201903878



Paso 2) Una vez que hayamos analizado el documento, imprimiremos el **"nombre de nodo"** de la raíz del documento y el "nombre de **etiqueta firstchild"** . El nombre de etiqueta y el nombre de nodo son las propiedades estándar del archivo XML.



- Importe el módulo xml.dom.minidom y declare el archivo que debe analizarse (myxml.xml)
- Este archivo contiene información básica sobre el empleado como nombre, apellido, casa, experiencia, etc.

- Usamos la función de análisis en el minidom XML para cargar y analizar el archivo XML
- Tenemos la variable doc y doc obtiene el resultado de la función de análisis
- Queremos imprimir el nombre de nodo y el nombre de etiqueta secundario del archivo, por lo que lo declaramos en la función de impresión
- Ejecute el código: imprime el nombre de nodo (#documento) del archivo XML y el primer nombre de etiqueta secundario (empleado) del archivo XML

Nota :

El nombre de nodo y el nombre de etiqueta hijo son los nombres o propiedades estándar de un dominio XML. En caso de que no esté familiarizado con este tipo de convenciones de nomenclatura.

Paso 3) También podemos llamar a la lista de etiquetas XML del documento XML e imprimirla. Aquí imprimimos el conjunto de habilidades como SQL, Python, Testing y Business.

- Declarar la pericia variable, de la cual vamos a extraer toda la pericia que tiene el empleado.
- Utilice la función estándar dom llamada "getElementsByTagName"
- Esto obtendrá todos los elementos llamados habilidad
- Declarar bucle sobre cada una de las etiquetas de habilidad
- Ejecute el código: le dará una lista de cuatro habilidades

IPC2
201903878

Cómo crear un nodo XML

Podemos crear un nuevo atributo usando la función "createElement" y luego agregar este nuevo atributo o etiqueta a las etiquetas XML existentes. Agregamos una nueva etiqueta "BigData" en nuestro archivo XML.

1. Debe codificar para agregar el nuevo atributo (BigData) a la etiqueta XML existente
2. Luego, debe imprimir la etiqueta XML con los nuevos atributos adjuntos a la etiqueta XML existente

- Para agregar un nuevo XML y agregarlo al documento, usamos el código "doc.create elementos"
- Este código creará una nueva etiqueta de habilidad para nuestro nuevo atributo "Big-data"
- Agregue esta etiqueta de habilidad en el documento primer hijo (empleado)
- Ejecute el código: aparecerá la nueva etiqueta "big data" con la otra lista de conocimientos

IPC2
201903878

EJEMPLOS

Ejemplo 1

Leer archivo XML e imprimir sus valores a través de Python (xml.dom.minidom).

Archivo de entrada

```
--staff.xml--

<?xml version="1.0"?>
<company>
    <name>Mkyong Enterprise</name>
    <staff id="1001">
        <nickname>mkyong</nickname>
        <salary>100,000</salary>
    </staff>
    <staff id="1002">
        <nickname>yflow</nickname>
        <salary>200,000</salary>
    </staff>
    <staff id="1003">
        <nickname>alex</nickname>
        <salary>20,000</salary>
    </staff>
</company>
```

Archivo.py

```
#dom-example.py

from xml.dom import minidom

doc = minidom.parse("staff.xml")

# doc.getElementsByTagName returns NodeList
name = doc.getElementsByTagName("name")[0]
print(name.firstChild.data)

staffs = doc.getElementsByTagName("staff")
for staff in staffs:
        sid = staff.getAttribute("id")
        nickname = staff.getElementsByTagName("nickname")[0]
        salary = staff.getElementsByTagName("salary")[0]
        print("id:%s, nickname:%s, salary:%s" %
              (sid, nickname.firstChild.data, salary.firstChild.data))
```

Salida

```
Mkyong Enterprise
id:1001, nickname:mkyong, salary:100,000
id:1002, nickname:yflow, salary:200,000
id:1003, nickname:alex, salary:20,000
```

Ejemplo 2

Utilizando el mismo archivo .xml de entrada el ejemplo 1

Archivo.py

```
from xml.dom import minidom

doc = minidom.parse("staff.xml")

def getNodeText(node):

    nodelist = node.childNodes
    result = []
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            result.append(node.data)
    return ''.join(result)

name = doc.getElementsByTagName("name")[0]
print("Node Name : %s" % name.nodeName)
print("Node Value : %s \n" % getNodeText(name))


staffs = doc.getElementsByTagName("staff")
for staff in staffs:
        sid = staff.getAttribute("id")
        nickname = staff.getElementsByTagName("nickname")[0]
        salary = staff.getElementsByTagName("salary")[0]
        print("id:%s, nickname:%s, salary:%s" %
              (sid, getNodeText(nickname), getNodeText(salary)))
```

Salida

```
Node Name : name
Node Value : Mkyong Enterprise
id:1001, nickname:mkyong, salary:100,000
id:1002, nickname:yflow, salary:200,000
id:1003, nickname:alex, salary:20,000
```

IPC2
201903878

Ejemplo 3

Analizando un XML

```
import xml.dom.minidom

def main():
    # use la función parse () para cargar y analizar un archivo XML
    doc = xml.dom.minidom.parse("Myxml.xml")

     # imprima el nodo del documento y el nombre de la primera etiqueta secu
ndaria
    print (doc.nodeName)
    print (doc.firstChild.tagName)
      # obtenga una lista de etiquetas XML del documento e imprima cada una
    expertise = doc.getElementsByTagName("expertise")
    print ("%d expertise:" % expertise.length)
    for skill in expertise:
        print (skill.getAttribute("name"))

    # Nueva etiqueta añadida
    newexpertise = doc.createElement("expertise")
    newexpertise.setAttribute("name", "BigData")
    doc.firstChild.appendChild(newexpertise)
    print (" ")

    expertise = doc.getElementsByTagName("expertise")
    print ("%d expertise:" % expertise.length)
    for skill in expertise:
        print (skill.getAttribute("name"))

if __name__ == "__main__":
    main()
```

Ejemplo 4

Archivo de entrada (ítems.xml)

```
<data>
<items>
        <item name="item1">item1abc</item>
        <item name="item2">item2abc</item>
    </items>
</data>
```

Archivo.py

```
from xml.dom import minidom

# parse an xml file by name
mydoc = minidom.parse('items.xml')

items = mydoc.getElementsByTagName('item')

# one specific item attribute
print('Item #2 attribute:')
print(items[1].attributes['name'].value)

# all item attributes
print('\nAll attributes:')
for elem in items:
    print(elem.attributes['name'].value)

# one specific item's data
print('\nItem #2 data:')
print(items[1].firstChild.data)
print(items[1].childNodes[0].data)

# all items data
print('\nAll item data:')
for elem in items:
    print(elem.firstChild.data)
```

Salida

```
$ python minidomparser.py
Item #2 attribute:
item2

All attributes:
```

```
item1
item2

Item #2 data:
item2abc
item2abc

All item data:
item1abc
item2abc
```

Ejemplo 5

```
import xml.dom.minidom

document = """\
<slideshow>
<title>Demo slideshow</title>
<slide><title>Slide title</title>
<point>This is a demo</point>
<point>Of a program for processing slides</point>
</slide>

<slide><title>Another demo slide</title>
<point>It is important</point>
<point>To have more than</point>
<point>one slide</point>
</slide>
</slideshow>
"""

dom = xml.dom.minidom.parseString(document)

def getText(nodelist):
    rc = []
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc.append(node.data)
    return ''.join(rc)

def handleSlideshow(slideshow):
    print("<html>")
    handleSlideshowTitle(slideshow.getElementsByTagName("title")[0])
    slides = slideshow.getElementsByTagName("slide")
    handleToc(slides)
```

```
    handleSlides(slides)
    print("</html>")

def handleSlides(slides):
    for slide in slides:
        handleSlide(slide)

def handleSlide(slide):
    handleSlideTitle(slide.getElementsByTagName("title")[0])
    handlePoints(slide.getElementsByTagName("point"))

def handleSlideshowTitle(title):
    print("<title>%s</title>" % getText(title.childNodes))

def handleSlideTitle(title):
    print("<h2>%s</h2>" % getText(title.childNodes))

def handlePoints(points):
    print("<ul>")
    for point in points:
        handlePoint(point)
    print("</ul>")

def handlePoint(point):
    print("<li>%s</li>" % getText(point.childNodes))

def handleToc(slides):
    for slide in slides:
        title = slide.getElementsByTagName("title")[0]
        print("<p>%s</p>" % getText(title.childNodes))

handleSlideshow(dom)
```

IPC2
201903878

XPATH (MÓDULO PYTHON)

Xpath es un módulo que es parte de la librería xml.etree.ElementTree por lo general la misma se importa de la siguiente manera:

```
import xml.etree.ElementTree as ET
```

Xpath provee una serie de expresiones para localizar elementos en un árbol, su finalidad es proporcionar un conjunto de sintaxis, por lo que debido a su limitado alcance no se considera un motor en si mismo.

Ejemplo:

```
import xml.etree.ElementTree as ET

root = ET.fromstring(docxml)

# Elementos de nivel superior
root.findall(".")

# todos los hijos de neighbor o nietos de country en el nivel superior
root.findall("./country/neighbor")

# Nodos xml con name='Singapore' que sean hijos de 'year'
root.findall(".//year/..[@name='Singapore']")

# nodos 'year' que son hijos de etiquetas xml cuyo name='Singapore'
root.findall(".//*[@name='Singapore']/year")

# todos los nodos 'neighbor'que son el segundo hijo de su padre
root.findall(".//neighbor[2]")
```

Como vemos nos permite extraer facilmente partes del xml haciendo referencia a su ubicación nodal representada a forma de path, lo que nos hace una sintaxis familiarmente sencilla a la hora de construir un paser xml.

SINTAXIS	Descripción
tag	Selecciona todos los elementos hijos contenidos en la etiqueta “tag”, Por ejemplo: spam, selecciona todos los elementos hijos de la etiqueta spam y así sucesivamente en un path de nodos spam/egg, /spam/egg/milk
*	Selecciona todos los elementos hijos. Ejemplo: */egg, seleccionara todos los elementos nietos bajo la etiqueta egg

.	Selecciona el nodo actual, este es muy usado en el inicio del path, para indicar que es un path relativo.
//	Selecciona todos los sub elementos de todos los niveles bajo el nodo expresado. Por ejemplo: .//egg selecciona todos los elementos bajo egg a través de todo el arbol bajo la etiqueta
..	Selecciona el elemento padre
[@attrib]	Selecciona todos los elementos que contienen el atributo tras el “@”
[@attrib=’value’]	Seleccione todos los elementos para los cuales el atributo dado tenga un valor dado, el valor no puede contener comillas
[tag]	Selecciona todos los elementos que contienen una etiqueta hijo llamada tag. Solo los hijos inmediatos son admitidos
[tag=’text’]	Selecciona todos los elementos que tienen una etiqueta hijo llamada tag incluyendo descendientes que sean igual al texto dado
[position]	Selecciona todos los elementos que se encuentran en la posición dada. La posición puede contener un entero siendo 1 la primera posición, la expresión last( ) para la ultima, o la posición relativa con respecto a la ultima posición last( )-1

notas: las expresiones entre corchetes deben ser precedidas por un nombre de etiquta, un astertisco u otro comodín. Las referencias a position deben ser precedidas por una etiqueta xml válida.

IPC2
201903878

EJEMPLOS

Ejemplo 1

El xml a continuación se utilizará para este ejemplo

```
<Catalog>
   <Books>
       <Book id="1" price="7.95">
           <Title>Do Androids Dream of Electric Sheep?</Title>
           <Author>Philip K. Dick</Author>
       </Book>
       <Book id="5" price="5.95">
           <Title>The Colour of Magic</Title>
           <Author>Terry Pratchett</Author>
       </Book>
       <Book id="7" price="6.95">
           <Title>The Eye of The World</Title>
           <Author>Robert Jordan</Author>
       </Book>
   </Books>
</Catalog>
```

Buscando todos los libros:

```
import xml.etree.cElementTree as ET
tree = ET.parse('sample.xml')
tree.findall('Books/Book')
```

Buscando el libro con título = 'El color de la magia':

```
tree.find("Books/Book[Title='The Colour of Magic']")
# always use '' in the right side of the comparison
```

Buscando el libro con id = 5:

```
tree.find("Books/Book[@id='5']")
# searches with xml attributes must have '@' before the name
```

Busque el segundo libro:

```
tree.find("Books/Book[2]")
# indexes starts at 1, not 0
```

Buscar el último libro:

```
tree.find("Books/Book[last()]")
# 'last' is the only xpath function allowed in ElementTree
```

Buscar todos los autores:

```
tree.findall(".//Author")
#searches with // must use a relative path
```

Ejemplo 2:

Leyendo un archivo xml

```
<data>
<items>
        <item name="item1">item1abc</item>
        <item name="item2">item2abc</item>
    </items>
</data>
```

Archivo.py

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

# one specific item attribute
print('Item #2 attribute:')
print(root[0][1].attrib)

# all item attributes
print('\nAll attributes:')
for elem in root:
    for subelem in elem:
        print(subelem.attrib)

# one specific item's data
print('\nItem #2 data:')
print(root[0][1].text)

# all items data
print('\nAll item data:')
```

```
for elem in root:
    for subelem in elem:
        print(subelem.text)
```

Salida

```
$ python treeparser.py
Item #2 attribute:
item2

All attributes:
item1
item2

Item #2 data:
item2abc

All item data:
item1abc
item2abc
```

Ejemplo 3

Contando los elementos de un archivo xml.

```
<data>
<items>
        <item name="item1">item1abc</item>
        <item name="item2">item2abc</item>
    </items>
</data>
```

Archivo.py

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

# total amount of items
print(len(root[0]))
```

Salida

```
$ python counterxml.py
2
```

Ejemplo 4

Escribir documentos XML

Haciendo uso del archivo xml (ítems.xml)

```
<data>
<items>
        <item name="item1">item1abc</item>
        <item name="item2">item2abc</item>
    </items>
</data>
```

Código ejemplo

```
import xml.etree.ElementTree as ET

# create the file structure
data = ET.Element('data')
items = ET.SubElement(data, 'items')
item1 = ET.SubElement(items, 'item')
item2 = ET.SubElement(items, 'item')
item1.set('name','item1')
item2.set('name','item2')
item1.text = 'item1abc'
item2.text = 'item2abc'

# create a new XML file with the results
mydata = ET.tostring(data)
myfile = open("items2.xml", "w")
myfile.write(mydata)
```

La ejecución de este código dará como resultado un nuevo archivo, "items2.xml", que debería ser equivalente al archivo "items.xml" original, al menos en términos de la estructura de datos XML.

Ejemplo 5:

Modificando elementos XML

Utilizando el archivo ítems.xml

```
<data>
<items>
        <item name="item1">item1abc</item>
        <item name="item2">item2abc</item>
    </items>
</data>
```

Código ejemplo:

```
import xml.etree.ElementTree as ET

tree = ET.parse('items.xml')
root = tree.getroot()

# changing a field text
for elem in root.iter('item'):
    elem.text = 'new text'

# modifying an attribute
for elem in root.iter('item'):
    elem.set('name', 'newitem')

# adding an attribute
for elem in root.iter('item'):
    elem.set('name2', 'newitem2')

tree.write('newitems.xml')
```

Después de ejecutar el código, el archivo XML resultante "newitems.xml" tendrá un árbol XML con los siguientes datos:

```
<data>
    <items>
        <item name="newitem" name2="newitem2">new text</item>
        <item name="newitem" name2="newitem2">new text</item>
    </items>
</data>
```

Como podemos ver al comparar con el archivo XML original, los nombres de los elementos del elemento han cambiado a "nuevo elemento", el texto a "nuevo texto" y el atributo "nombre2" se ha agregado a ambos nodos.