



Modelo DOM - Procesamiento XML

Laboratorio IPC2



Qué es DOM?

DOM o Document Object Model es un lenguaje para acceder y modificar documentos XML.


Con el Modelo de Objetos del Documento cualquier desarrollador puede construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido.

El mejor ejemplo del modelo DOM es HTML.



Ejemplo del modelo DOM en archivo XML

```
<zoo>
  <mamiferos>
    <animal> Gato </animal>
    <animal> Perro </animal>
  </mamiferos>
  <reptiles>
    <animal> Boa </animal>
    <animal> Lagarto </animal>
  </reptiles>
</zoo>
```

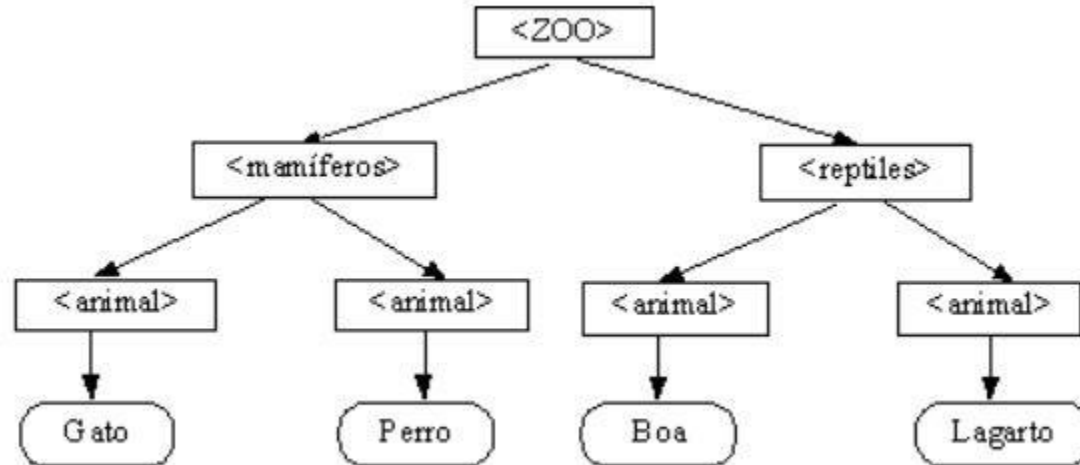


En el DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol sin embargo, no especifica que debe ser implementado como un árbol obligatoriamente.

El DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente.

Una propiedad importante de los modelos de estructura del DOM es algo llamado “*isomorfismo estructural*”, el cual significa que si dos implementaciones cualesquiera del Modelo de Objetos del Documento se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

Representación gráfica del ejemplo anterior






Origen del Modelo DOM

El DOM se originó como una especificación para permitir que los programas Java y los scripts de JavaScript fueran portables entre los navegadores web.

La primera especificación del modelo surgió en 1997 llamada DOM Level 1 del cual se realizó una actualización en el año 2000.

Se han realizado nuevas especificaciones al modelo agregando más funcionalidades para el procesamiento de archivos con propiedades que necesitan de otras funciones.



Los elementos que componen al archivo XML son tratados por el modelo DOM como objetos. Los atributos que contienen estos objetos son tratados por el modelo DOM como cadenas de caracteres o strings, por lo que tenemos que tomarlo en cuenta al realizar operaciones con los datos recién obtenidos del archivo de entrada.

Los atributos en el modelo DOM también pueden ser tratados como nodos en vez de strings aunque es bastante inusual hacerlo.



Objetos en el modelo DOM

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 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.




MiniDOM

MiniDOM o Minimal DOM Implementation es una implementación simplificada del modelo DOM que puede utilizarse en Python.

Para utilizarlo es necesario importarlo desde el módulo `xml.dom`.

Este modelo usa la función “`parse`” para crear un objeto DOM directamente desde nuestro archivo XML. La función tiene la siguiente sintaxis:

```
xml.dom.minidom.parse(filename_or_file[, parser[, bufsize]])
```




La propiedad "file_or_filename" debe contener una ruta hacia el archivo XML o hacia un objeto de tipo archivo. Esta función retorna un documento el cual ya podremos procesar como un XML.

Para empezar a buscar elementos con el nombre específico de una tag utilizamos el siguiente comando: **getElementByTagName()**.

Debido a que cada nodo se trata como un objeto, podemos acceder a los valores de los tags usando las propiedades del objeto.

En el ejemplo siguiente podemos observar el procesamiento de un archivo XML.



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



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

```
mydoc = minidom.parse('items.xml')
```

Primero debemos importar la librería y mediante el comando “parse” creamos un objeto documento con la ruta del archivo XML.

```
items = mydoc.getElementsByTagName('item')
```

Para obtener los nodos con el tag “item” utilizamos el comando “getElementsByTagName”.



Podemos obtener los valores de los atributos del tag de la siguiente manera:


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

Con lo cual tendremos de resultado:

```
Item #2 attribute:  
item2
```

Para imprimir todos los valores posibles de los atributos podemos usar instrucciones cíclicas.

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



Lo cual nos dará el siguiente resultado:


```
All attributes:  
item1  
item2
```

Para obtener los valores dentro del tag tenemos dos opciones. Utilizar el comando "firstChild" o utilizar `childNodes[x]` donde `x` es el índice. Usualmente este índice será 0.

```
print('\nItem #2 data:')  
print(items[1].firstChild.data)  
print(items[1].childNodes[0].data)
```

De cualquier forma, el resultado será el mismo:

```
Item #2 data:  
item2abc  
item2abc
```



Ahora bien, si lo que desea es obtener todos los valores posibles dentro del tag especificado, también podemos usar instrucciones cíclicas de la siguiente manera:

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

El resultado será el siguiente:

```
All item data:  
item1abc  
item2abc
```





Element Tree

Es otra alternativa simple para procesar archivos XML que se puede utilizar en Python. Igual que en MiniDOM, debemos primero importar la librería para utilizar el módulo.

Al utilizar Element Tree, se crea una estructura en forma de árbol utilizando el comando “parse” y se obtiene un elemento raíz. Una vez identificado este elemento, ya podremos recorrer el árbol debido a que todos los nodos del árbol están conectados.

Utilizaremos el mismo archivo de ejemplo que en MiniDOM para ejemplificar algunas funciones básicas de Element Tree.



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

Primero debemos importar la librería y mediante el comando “parse” creamos un objeto de tipo árbol. Con el comando “getroot()” obtenemos la raíz.

En este caso, con el Element Tree no utilizaremos cadenas para buscar los tags específicos, sino que utilizaremos una notación en la que especificamos índices de subárboles y posiciones dentro de sus nodos.

Utilizaremos el comando “root[X][Y].attribute”.

Donde **X** es el índice del nodo padre que buscamos y **Y** será el índice del subnodo.




Para buscar un elemento en especial:

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

Obtenemos el siguiente resultado:

```
Item #2 attribute:  
item2
```

Para obtener todos los valores de cierto atributo también utilizamos instrucciones cíclicas con la característica de que utilizamos los identificadores `root` para la raíz, `element` para los atributos en tags y `subelement` para los valores.



```
print('\nAll attributes:')
for elem in root:
    for subelem in elem:
        print(subelem.attrib)
```

Y obtenemos el siguiente resultado:

```
All attributes:
item1
item2
```

Para obtener un valor específico:

```
print('\nItem #2 data:')
print(root[0][1].text)
```



El resultado será:

```
Item #2 data:  
item2abc
```

Para obtener todos los valores de las tags

```
print('\nAll item data:')  
for elem in root:  
    for subelem in elem:  
        print(subelem.text)
```

Y el resultado será:

```
All item data:  
item1abc  
item2abc
```



CONTANDO ELEMENTOS DE UN DOCUMENTO XML (USANDO DOM)

minidom se debe importar desde el modulo dom.

En este modulo se encuentra la funcion `getElementsByTagName`,

que usaremos para encontrar el elemento de etiqueta.

Tenga en cuenta que esto solo contará la cantidad de elementos secundarios debajo de la nota en la que ejecuta `len()`, que en este caso es el nodo raíz. Si desea encontrar todos los subelementos en un árbol mucho más grande, deberá recorrer todos los elementos y contar cada uno de sus elementos secundarios.

```
from xml.dom import minidom

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

items = mydoc.getElementsByTagName('item')

# total amount of items
print(len(items))
```



CONTANDO ELEMENTOS USANDO (ElementTree)

Del mismo modo, el ElementTree módulo nos permite calcular la cantidad de nodos conectados a un nodo.

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

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

ESCRIBIR DOCUMENTOS XML(ElementTree)

ElementTree también es ideal para escribir datos en archivos XML. El siguiente código muestra cómo crear un archivo XML con la misma estructura que el archivo que usamos en los ejemplos anteriores.


1. Cree un elemento, que actuará como nuestro elemento raíz. En este caso se llamará "data"
2. Una vez que tenemos nuestro elemento raíz, podemos crear subelementos usando la SubElement función sintaxis:

```
import xml.etree.ElementTree as ET

# create the file structure
data = ET.Element('data')
```

```
items = ET.SubElement(data, 'items')
```


```
SubElement(parent, tag, attrib={}, **extra)
```

Aquí parent está el nodo principal al que conectarse, attrib hay un diccionario que contiene los atributos del elemento y extra hay argumentos de palabras clave adicionales. Esta función nos devuelve un elemento, que puede usarse para adjuntar otros subelementos, como lo hacemos en las siguientes líneas al pasar elementos al SubElement constructor.

3. Aunque podemos agregar nuestros atributos con la SubElement función, también podemos usar la set() función, como hacemos en el siguiente código. El texto del elemento se crea con la propiedad text del Elemento objeto.

```
item1 = ET.SubElement(items, 'item')
item2 = ET.SubElement(items, 'item')
item1.set('name', 'item1')
item2.set('name', 'item2')
item1.text = 'item1abc'
item2.text = 'item2abc'
```



4. En las últimas 3 líneas del código siguiente, creamos una cadena a partir del árbol XML y escribimos esos datos en un archivo que abrimos.

```
# 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. Probablemente notará que la cadena resultante es solo una línea y no contiene sangría,



ENCONTRAR ELEMENTOS EN XML


El ElementTree módulo ofrece la findall() función, que nos ayuda a encontrar elementos específicos en el árbol. Devuelve todos los artículos con la condición especificada. Además, el módulo tiene la función find(), que devuelve solo el primer subelemento que coincide con los criterios especificados.

Sintaxis:

```
findall(match, namespaces=None)
```

```
find(match, namespaces=None)
```

Para ambas funciones, el match parámetro puede ser un nombre de etiqueta XML o una ruta. La función findall() devuelve una lista de elementos y find devuelve un solo objeto de tipo Element.



se recorren todos los elementos de l root y todos los subElementos 'item'

1. Obtenemos el dict de cada elemento para obtener el nombre del atributo en este caso 'name'
2. Obtenemos únicamente el nombre de cada elemento si ya conocemos el atributo nombrado 'name'

```
for elem in root:
    for subelem in elem.findall('item'):

        # if we don't need to know the name of the attribute
        print(subelem.attrib)

        # if we know the name of the attribute
        print(subelem.get('name'))
```



```
{'name': 'item1'}
item1
{'name': 'item2'}
item2
```



MODIFICAR UN ELEMENTO

El ElementTree módulo presenta varias herramientas para modificar documentos XML existentes. El siguiente ejemplo muestra cómo cambiar el nombre de un nodo, cambiar el nombre de un atributo y modificar su valor, y cómo agregar un atributo adicional a un elemento.

El texto de un nodo se puede cambiar especificando el nuevo valor en el campo de texto del objeto de nodo. El nombre del atributo se puede redefinir utilizando la `set(name, value)` función. La `set` función no tiene que trabajar solo en un atributo existente, también se puede usar para definir un nuevo atributo.

```
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')
```

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





ELIMINAR UN ELEMENTO XML

Como probablemente esperarías, el `ElementTree` módulo tiene la funcionalidad necesaria para eliminar los atributos y subelementos del nodo.


El siguiente código muestra cómo eliminar el atributo de un nodo utilizando la `pop()` función. La función se aplica al `attrib` parámetro de objeto. Especifica el nombre del atributo y lo establece en `None`.

```
import xml.etree.ElementTree as ET

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

# removing an attribute
root[0][0].attrib.pop('name', None)

# create a new XML file with the results
tree.write('newitems3.xml')
```



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

Como se observa se eliminó el atributo name del primer subElemento



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


ELIMINAR SUB-ELEMENTO

Un subelemento específico se puede eliminar usando la remove función. Esta función debe especificar el nodo que queremos eliminar.

```
import xml.etree.ElementTree as ET

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

# removing one sub-element
root[0].remove(root[0][0])

# create a new XML file with the results
tree.write('newitems4.xml')
```



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

ELIMINAR TODOS LOS ELEMENTOS

El ElementTree módulo nos presenta la `clear()` función, que se puede usar para eliminar todos los subelementos de un elemento dado.

El siguiente ejemplo nos muestra cómo utilizar `clear()`:

```
import xml.etree.ElementTree as ET

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

# removing all sub-elements of an element
root[0].clear()

# create a new XML file with the results
tree.write('newitems5.xml')
```



```
<data>
  <items />
</data>
```