DOM xml

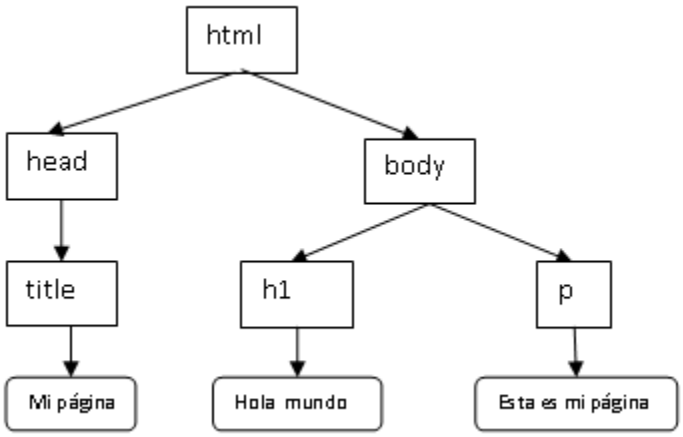
El Modelo de Objetos del Documento, o «DOM» por sus siglas en inglés, es un lenguaje API del Consorcio *World Wide Web* (W3C) para acceder y modificar documentos XML. Una implementación del DOM presenta los documento XML como un árbol, o permite al código cliente construir dichas estructuras desde cero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

En el DOM los documentos tienen una estructura parecida a un arbol, creando una estructura jerarquica en la que de un objeto principal pueden depender varios secundarios. Esto se ve claramente en la escructura que tiene la página de HTML, la cual consta de etiquetas anidadas.

Por ejemplo si tenemos una página web sencilla:

```
<html>
<head>
  <title>mi página</title>
</head>
<body>
  <h1>Hola mundo</h1>
        <p>Esta es mi página.</p>
</body>
</html>
```

En el DOM se representa como una estructura de arbol de la siguiente manera:



Cada uno de los recuadros de la estructura anterior es un **nodo**. Los nodos son los elementos básicos de la estructura del DOM.

Tipos de nodos

Aunque existen 12 tipos de nodos en realidad en las páginas web sólo tenemos los 5 siguientes:

- **Document:** nodo raiz del que derivan todos los demas.
- **Element:** Cada una de las etiquetas HTML. Es el unico nodo que puede contener atributos y del que pueden derivar otros nodos.
- **Attr;** Cada atributo de una etiqueta genera un nodo Attr, el cual contiene también su valor (como una propiedad). Es hijo del nodo element (etiqueta) que lo contiene.
- **Text:** Contiene el texto encerrado por una etiqueta HTML (hijo del nodo Element).
- **Comment:** Los comentarios incluidos en la pagina HTML también generan sus nodos.

El Modelo de Objetos del Documento es definido por el *W3C* en fases, o «niveles» en su terminología. El mapeado de Python de la API está basado en la recomendación del DOM nivel 2.

Ejemplos de Procesamiento de XML Python

Python utiliza el modelo de objetos de documento para almacenar y manipular elementos XML. Se analiza automáticamente el código XML y crea un objeto de documento para modelarlo, que contiene nodos XML. La biblioteca XML a continuación, puede desplazarse por el documento de encontrar diferentes elementos y atributos. Debe importar la biblioteca "xml.dom.minidom" con el fin de acceder a la biblioteca de procesamiento de XML DOM de Python.

Acceder y leer un archivo XML

biblioteca de Python XML puede analizar automáticamente los archivos de texto XML o cadenas de texto que representan un documento XML. Aquí está un ejemplo de cómo analizar un archivo XML de texto y guardar el resultado como un objeto de documento de Python:

```
xmlFile = open ( "sample.xml")
xmlDocument = xml.dom.minidom.parse (xmlFile)
```

Acceso a Element nodos y nodos secundarios

elementos XML se representan como nodos. Para acceder a un elemento en el documento XML, debe buscar por su nombre con el método "getElementsByTagName", por ejemplo:

searchResults = getElementsByTagName ("ExampleNode")

Esta línea de código devuelve un NodeList. Para obtener un único nodo elemento, sólo tiene que llamar:

miNodo = searchResults [0]

Los nodos pueden contener nodos secundarios. Usted puede obtener una lista de nodos secundarios dado una referencia a un nodo padre, por ejemplo:

myChildNodes = myNode.childNodes

Datos y atributos

Para acceder a los datos contenidos en las etiquetas de apertura y cierre de un elemento, debe tener acceso al campo "datos" del objeto de nodo. Por ejemplo, si un nodo denominado "miNodo" representado el texto XML "<ExampleNode attr1 =" 1 "attr2 =" 2 "> sampletext </ ExampleNode>" en un documento, entonces se podría extraer la palabra "sampletext" haciendo referencia a " myNode.data ". Para acceder a los atributos de ese nodo, lo que se necesita para acceder a la primera NamedNodeMap de los atributos de la siguiente manera:

listadeatributos = myNode.attributes

Luego de esa lista, de extraer los nombres y valores de los atributos:

```
myList = []
for i in range (attrList.length): myList.append (attrList.item (i) .name + '=' + attrList.item (i) .value)
print ";" .join (miLista)
```

Escribir código XML

Se puede escribir un objeto de nodo a cualquier objeto "puede escribir" usando la función "WriteXml." Esto incluye los permisos de escritura. Un ejemplo es el siguiente:

```
destinationFile = open ( "samplewrite.xml", "w")
doc.writexml (destinationFile)
destinationFile.close ()
```

También puede imprimir el documento XML como una cadena utilizando la función "toxml" o "toprettyxml", por ejemplo:

doc.toxml de impresión ()

o

doc.toprettyxml de impresión ()

La función "toprettyxml" utiliza espacios y guiones para hacer más legible el código XML para los seres humanos.

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.

Ejemplos de uso de Xpath:

```
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.

Xpath Sintaxis:

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.

Ejemplo de uso de Xpath dentro de una sentencia xml:

```
1    <xpath expr="//field[@name]='is_done'" position="before">
2     <field name="date_deadline" />
3    </xpath>
```

Analizar un documento XML

Los documentos XML analizados se representan en la memoria mediante objetos ElementTree Element conectados en una estructura de árbol basada en la forma en que los nodos están anidados en el documento XML.

Analizar un documento completo con parse() devuelve una instancia ElementTree. El árbol conoce todos los datos en el documento de entrada, y los nodos del árbol se pueden buscar o manipulados en su lugar. Si bien esta flexibilidad puede hacer que trabajar con el documento analizado sea más conveniente, normalmente requiere más memoria que un enfoque de análisis basado en eventos ya que todo el documento debe ser cargado a la vez.

La huella de memoria de documentos pequeños y simples como esta lista de podcasts representada como un esquema OPML no son significativa:

```
<?xml version="1.0" encoding="UTF-8"?>
<opml version="1.0">
<head>
    <title>My Podcasts</title>
    <dateCreated>Sat, 06 Aug 2016 15:53:26 GMT</dateCreated>
    <dateModified>Sat, 06 Aug 2016 15:53:26 GMT</dateModified>
</head>
<body>
  <outline text="Non-tech">
    <outline
        text="99% Invisible" type="rss"

xmlUrl="http://feeds.99percentinvisible.org/99percentinvisible"
        htmlUrl="http://99percentinvisible.org" />
  </outline>
  <outline text="Python">
    <outline
        text="Talk Python to Me" type="rss"
        xmlUrl="https://talkpython.fm/episodes/rss"
        htmlUrl="https://talkpython.fm" />
```

```
  <outline
        text="Podcast.__init__" type="rss"
        xmlUrl="http://podcastinit.podbean.com/feed/"
        htmlUrl="http://podcastinit.com" />
  </outline>
</body>
</opml>
```

Para analizar el archivo, pasa un identificador de archivo abierto a `parse()`.

ElementTree_parse_opml.py

```
from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:
    tree = ElementTree.parse(f)

print(tree)
```

Leerá los datos, analizará el XML y devolverá un objeto `ElementTree`.

```
$ python3 ElementTree_parse_opml.py

<xml.etree.ElementTree.ElementTree object at 0x1013e5630>
```

Atravesar el árbol analizado

Para visitar a todos los niños en orden, usa iter() para crear un generador que itera sobre la instancia ElementTree.

ElementTree_dump_opml.py

```
from xml.etree import ElementTree
import pprint

with open('podcasts.opml', 'rt') as f:
    tree = ElementTree.parse(f)

for node in tree.iter():
    print(node.tag)
```

Este ejemplo imprime todo el árbol, una etiqueta a la vez.

```
$ python3 ElementTree_dump_opml.py

opml
head
title
dateCreated
dateModified
body
outline
outline
outline
outline
outline
```

Para imprimir solo los grupos de nombres y URL de fuentes para los podcasts, dejando fuera de todos los datos en la sección de encabezado iterando sobre solo los nodos outline e imprimir los atributos text y xmlUrl buscando los valores en el diccionario **attrib**.

ElementTree_show_feed_urls.py

```
from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:
    tree = ElementTree.parse(f)

for node in tree.iter('outline'):
    name = node.attrib.get('text')
    url = node.attrib.get('xmlUrl')
    if name and url:
        print('  %s' % name)
        print('    %s' % url)
    else:
        print(name)
```

El argumento 'outline' para iter() significa que el procesamiento esa limitado solo a nodos con la etiqueta 'outline'.

```
$ python3 ElementTree_show_feed_urls.py

Non-tech
  99% Invisible
    http://feeds.99percentinvisible.org/99percentinvisible
Python
  Talk Python to Me
    https://talkpython.fm/episodes/rss
  Podcast.__init__
    http://podcastinit.podbean.com/feed/
```

Encontrar nodos en un documento

Caminar por todo el árbol de esta manera, buscando nodos relevantes, puede ser propenso a errores. El ejemplo anterior tenía que mirar cada nodo outline para determinar si era un grupo (solo nodos con un atributo **text**) o podcast (con ambos **text** y **xmlUrl**). Para producir una lista simple de las URL de los canales de información de podcast, sin nombres o grupos, la lógica podría simplificarse utilizando findall() para buscar nodos con características de búsqueda más descriptivos.

Como primer paso para convertir la primera versión, se puede utilizar un argumento XPath para buscar todos los nodos outline.

ElementTree_find_feeds_by_tag.py

```
from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:
    tree = ElementTree.parse(f)

for node in tree.findall('.//outline'):
    url = node.attrib.get('xmlUrl')
    if url:
        print(url)
```

La lógica en esta versión no es sustancialmente diferente a la versión utilizando getiterator(). Todavía tiene que comprobar la presencia de la URL, excepto que no imprime el nombre del grupo cuando no se encuentra la URL.

```
$ python3 ElementTree_find_feeds_by_tag.py

http://feeds.99percentinvisible.org/99percentinvisible
https://talkpython.fm/episodes/rss
http://podcastinit.podbean.com/feed/
```

Es posible aprovechar el hecho de que los nodos outline solo se anidan dos niveles de profundidad. Cambiando la ruta de búsqueda a .//outline/outline significa que el bucle procesará solo el segundo nivel de nodos outline.

ElementTree_find_feeds_by_structure.py

```
from xml.etree import ElementTree

with open('podcasts.opml', 'rt') as f:
    tree = ElementTree.parse(f)

for node in tree.findall('.//outline/outline'):
    url = node.attrib.get('xmlUrl')
    print(url)
```

Se espera que todos los nodos outline anidados dos niveles de profundidad en la entrada tengan el atributo xmlURL que se refiere a la fuente de podcast, por lo que el bucle puede omitir la comprobación del atributo antes de usarlo.

```
$ python3 ElementTree_find_feeds_by_structure.py

http://feeds.99percentinvisible.org/99percentinvisible
https://talkpython.fm/episodes/rss
http://podcastinit.podbean.com/feed/
```

Sin embargo, esta versión está limitada a la estructura existente, por lo que si los nodos outline se reorganizan en un árbol más profundo, dejará de trabajar.