¿Qué es XML?

XML son las siglas de eXtensible Markup Language. Fue diseñado para almacenar y transportar pequeñas o medianas cantidades de datos y se usa ampliamente para compartir información estructurada.

Python le permite analizar y modificar documentos XML. Para analizar un documento XML, debe tener todo el documento XML en la memoria.

xml.dom

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 documentos 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.

El DOM es extremadamente útil para aplicaciones de acceso directo

Algunas aplicaciones son imposibles en un modelo orientado a eventos sin acceso a un árbol. Por supuesto que puedes construir algún tipo de árbol por tu cuenta en eventos SAX, pero el DOM te evita escribir ese código. El DOM es una representación de árbol estándar para datos XML.

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.

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 métodos 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.

Una vez que tengas un objeto del documento del DOM, puedes acceder a las partes de tu documento XML a través de sus propiedades y métodos.

Contenido del Módulo

El módulo xml.dom contiene las siguientes funciones:

xml.dom.registerDOMImplementation(name, factory)

Registra la función factory con el nombre name. La función fábrica (factory) debe retornar un objeto que implemente la interfaz DOMImplementation. La función fábrica puede retornar el mismo objeto cada vez que se llame, o uno nuevo por cada llamada, según sea apropiado para la implementación específica (e.g. si la implementación soporta algunas personalizaciones).

xml.dom.getDOMImplementation(name=None, features=())

Retorna una implementación del DOM apropiada. El name es o bien conocido, el nombre del módulo de una implementación DOM, o None. Si no es None importa el módulo correspondiente y retorna un objecto DomImplementation si la importación tiene éxito. Si no se le pasa un nombre, y el entorno de variable PYTHON_DOM ha sido puesto, dicha variable es usada para encontrar la información de la implementación.

Si no se le pasa un nombre, examina las implementaciones disponibles para encontrar uno con el conjunto de características requeridas. Si no se encuentra ninguna implementación, levanta una excepción ImportError. La lista de características debe ser una secuencia de pares (feature,version) que son pasados al método hasFeature() en objetos disponibles de DOMImplementation.

Además, xml.dom contiene una clase base Node y las clases de excepciones del DOM. La clase Node proporcionada por este módulo no implementa ninguno de los métodos o atributos definidos en la especificación DOM; las implementaciones del DOM concretas deben definirlas. La clase Node propuesta por este módulo sí proporciona las constantes usadas por el atributo nodeType en objetos concretos de Node; estas son localizadas dentro de la clase en vez de estar al nivel del módulo para cumplir las especificaciones del DOM.

staff.xml

```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>
```

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
```

dom-example2.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
```

Qué es XPath

El lenguaje Xpath es el sistema que se utiliza para navegar y consultar los elementos y atributos contenidos en la estructura de un documento XML.

Para qué sirve Xpath

XPath sirve para que los programadores puedan definir criterios de búsqueda avanzada y cálculos específicos, utilizando una sintaxis simple, pero bastante eficaz.

Contiene una librería de 100 funciones estándar, que permiten realizar operaciones para el manejo de cadenas, operaciones numéricas, comparaciones de fechas… etcétera.

XPath no es un lenguaje independiente, pues se usa en combinación con XSLT (eXtensible Stylesheet Language for Transformations) o lenguaje que permite aplicar una transformación a un documento XML.

Ejemplos de Xpath

Para XPath, un documento XML es como un árbol, que está compuesto por dos conceptos. Por un lado, la representación de distintos tipos de nodos; por el otro, las posibles relaciones que existen entre estos nodos.

Existen varios tipos de nodos, los más básicos se denominan:

- Nodo Elemento.
- Nodo Atributo.
- Nodo Texto.
- Nodo raíz.
- Elemento raíz.
- Valores atómicos.

El nodo raíz del árbol contiene al elemento raíz del documento. Todos los documentos XML tienen un nodo raíz que indica el inicio del documento, de donde dependen todos los demás nodos del documento.

Ejemplo de uso de libxml2 XPath

```
import libxml2

doc = libxml2.parseFile("tst.xml")
ctxt = doc.xpathNewContext()
res = ctxt.xpathEval("//*")
if len(res) != 2:
    print "xpath query: wrong node set size"
    sys.exit(1)
if res[0].name != "doc" or res[1].name != "foo":
    print "xpath query: wrong node set value"
    sys.exit(1)
doc.freeDoc()
ctxt.xpathFreeContext()
```

Ejemplo de uso de ElementTree XPath

```
from elementtree.ElementTree import ElementTree
mydoc = ElementTree(file='tst.xml')
for e in mydoc.findall('/foo/bar'):
    print e.get('title').text
```

```
 <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
```