201906085 Sebastian Alejandro de Leon Tenaz

IPC2

ACTIVIDAD 2

DOM XML:

El Document Object Model ("DOM" para abreviar) es un lenguaje API del World Wide Web Consortium (W3C) para acceder y modificar documentos XML. Las implementaciones DOM representan documentos XML como árboles o permiten que el código del cliente construya estas estructuras desde cero y luego les permita acceder a las estructuras a través de un conjunto de objetos que implementan interfaces conocidas.

DOM es muy útil para aplicaciones de atajos. SAX solo le permite ver una parte del documento a la vez. Si desea ver un elemento SAX, no tiene permiso para acceder a otro elemento. Si está viendo un nodo de texto, no tiene acceso al elemento contenedor. Al desarrollar una aplicación SAX, necesita registrar la ubicación del programa en una determinada ubicación del código en el archivo. SAX no hará esto por ti. Además, lamentablemente, no podrá mirar hacia adelante en el documento XML.

Sin acceso al árbol, ciertas aplicaciones no son posibles en el modelo controlado por eventos. Por supuesto, usted mismo puede construir algún tipo de árbol en eventos SAX, pero el DOM le impide escribir ese código. DOM es una representación de árbol estándar de datos XML.

Una vez que tenga un objeto de documento DOM, puede acceder a varias partes del documento XML a través de sus propiedades y métodos. Estos atributos se definen en la especificación DOM. Esta parte del manual describe la interpretación de las especificaciones en Python.

EJEMPLO:

1.

```
from xml.dom import minidom

doc = minidom.parse("/ruta/datos.xml")

nombre = doc.getElementsByTagName("nombre")[0]

print(nombre.firstChild.data)

empleados = doc.getElementsByTagName("empleado")

for empleado in empleados:

sid = empleado.getAttribute("id")

username = empleado.getElementsByTagName("username")[0]

password = empleado.getElementsByTagName("password")[0]

print("id:%s " % sid)
```

```
print("username:%s" % username.firstChild.data)

print("password:%s" % password.firstChild.data)
```

2.

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


dom1 = parse('c:\\temp\\mydata.xml')  # parse an XML file by name


datasource = open('c:\\temp\\mydata.xml')

dom2 = parse(datasource)  # parse an open file


dom3 = parseString('<myxml>Some data<empty/> some more data</myxml>')
```

3.

```
from xml.dom.minidom import getDOMImplementation


impl = getDOMImplementation()


newdoc = impl.createDocument(None, "some_tag", None)

top_element = newdoc.documentElement

text = newdoc.createTextNode('Some textual content.')

top_element.appendChild(text)
```

xPath XML para python:

Tiene una serie de ventajas:

-Cumplimiento de la especificación

-Desarrollo activo y participación comunitaria

-Velocidad. Esta es realmente una envoltura de Python alrededor de una implementación de C.

-Ubicuidad. La biblioteca libxml2 es omnipresente y, por lo tanto, está bien probada.

Las desventajas incluyen:

-Cumplimiento de la especificación . Es estricto. Cosas como el manejo del espacio de nombres predeterminado son más fáciles en otras bibliotecas.

-Uso de código nativo. Esto puede ser una molestia dependiendo de cómo se distribuya / implemente su aplicación. Hay disponibles RPM que alivian algo de este dolor.

-Manejo manual de recursos. Observe en el ejemplo siguiente las llamadas a freeDoc () y xpathFreeContext (). Esto no es muy pitónico.

-Si está haciendo una selección de ruta simple, quédese con ElementTree (que se incluye en Python 2.5). Si necesita el cumplimiento total de las especificaciones o la velocidad bruta y puede hacer frente a la distribución de código nativo.

EJEMPLO:

1.

```
 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()
```

2.

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

```
print e.get('title').text
```