Universidad San Carlos de Guatemala

Laboratorio de introducción a la programación y computación 2

Sección E

Actividad 2

Federico Guillermo

Iturbide Cerna

201905699

06/02/2021

XML

XML es el acrónimo de Extensible Markup Language, es decir, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos.

El lenguaje de marcado es un conjunto de códigos que se pueden aplicar en el análisis de datos o la lectura de textos creados por computadoras o personas. El lenguaje XML proporciona una plataforma para definir elementos para crear un formato y generar un lenguaje personalizado.

Un archivo XML se divide en dos partes: prolog y body. La parte prolog consiste en metadatos administrativos, como declaración XML, instrucción de procesamiento opcional, declaración de tipo de documento y comentarios. La parte del body se compone de dos partes: estructural y de contenido (presente en los textos simples).

El diseño XML se centra en la simplicidad, la generalidad y la facilidad de uso y, por lo tanto, se utiliza para varios servicios web. Tanto es así que hay sistemas destinados a ayudar en la definición de lenguajes basados en XML, así como APIs que ayudan en el procesamiento de datos XML, que no deben confundirse con HTML.

DOM en python

En Python se utiliza “xml.dom.minidom”, lo que es una implementación con una api similar a la de otros lenguajes. Esta está destinada a ser mas simple que una implementación completa del DOM y también mas liviana, por lo que los usuarios que aun no dominen el DOM deberían considerar usar el módulo “xml.etree.ElementTree” para el procesamiento XML.

También se debe aclarar que el módulo minidom no es seguro contra datos maliciosos, por lo que se debe tener cuidado al procesar datos si no son de confianza o no han sido autenticados.

Ejemplo #1

En el primer ejemplo se parsea un archivo .xml, esto mediante las funciones de análisis sintáctico

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

dom1 = parse('c:\\temp\\mydata.xml') <-Convierte el archivo .xml

datasource = open('c:\\temp\\mydata.xml')
dom2 = parse(datasource) <-Convierte el archivo ya abierto

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

Ejemplo #2

En este segundo ejemplo se creará un archivo xml y una vez esté creado podemos empezar a añadir nodos secundarios

```
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)
```

Ejemplo #3

En este ejemplo tenemos un fragmento de una clase, donde por un método recorremos el archivo y sus nodos

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

Ejemplo #4

Este ejemplo nos muestra como se obtiene el archivo .xml, luego lo enlista y por último por un ciclo donde recorre lo guardado para ordenarlo e imprimirlo.

```
from xml.dom import minidom
doc = minidom.parse("staff.xml")
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))
```

Ejemplo #5

En este último ejemplo se lee el archivo xml y luego por medio de una función recorremos y validamos los nodos en nuestro archivo, luego los imprimimos y ahora recorremos los elementos y los ordenamos según id, nickname y salario.

```
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)))
```

xPath en Python

En este caso tenemos xPath, donse se utiliza “xml.etree.ElementTree” para nuestra creación de datos xml, donde también tenemos ciertas vulnerabilidades de datos asi como en DOM.

lxml.etree admite la sintaxis de ruta simple de los métodos find, findall y findtext en ElementTree y Element, como se conoce en la biblioteca ElementTree original (ElementPath), como extensión específica de lxml, estas clases también proporcionan un método xpath () que admite expresiones en la sintaxis completa de XPath, así como funciones de extensión personalizadas.

También hay clases de evaluador XPath especializadas que son más eficientes para evaluaciones frecuentes: XPath y XPathEvaluator. Se debe consultar la comparación de rendimiento para saber cuándo usar cuál. Su semántica cuando se usa en Elements y ElementTrees es la misma que para el método xpath ().

Teniendo en cuenta que los métodos .find * () suelen ser más rápidos que el soporte completo de XPath. También admiten el procesamiento de árbol incremental a través del método .iterfind (), mientras que XPath siempre recopila todos los resultados antes de devolverlos.

Ejemplo #1

Este ejemplo es bastante básico, importamos la librería, luego decimos que archivo deseamos leer y por medio del for recorremos el arreglo.

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

Ejemplo #2

En este tramo de código recorremos el archivo xml previamente guardado y verificamos su longitud para saber cómo ordenarlo.

```
def reorder_attributes(root):
    for el in root.iter():
        attrib = el.attrib
        if len(attrib) > 1:
            # adjust attribute order, e.g. by sorting
            attribs = sorted(attrib.items())
            attrib.clear()
            attrib.update(attribs)
```

Ejemplo #3

En este ejemplo se puede observar como se procesan los datos y luego se compara el tamaño en el primer if, luego comparamos que las columnas sean iguales a las deseadas y si es el caso procesamos los datos.

```
import xml.etree.ElementTree as ET

doc = ET.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()
```