Nombre: David Alejandro Hernández de León
Carné: 201700650
Curso: Laboratorio IPC2
Sección: E

INVESTIGACIÓN

DOM

Es una implementación mínima de la interfaz Document Object Model (Modelo de objetos del documento), con una API similar a la de otros lenguajes. Está destinada a ser más simple que una implementación completa del DOM y también significativamente más pequeña. Aquellos usuarios que aún no dominen el DOM deberían considerar usar el módulo xml.etree.ElementTree en su lugar para su procesamiento XML.

- Se accede a las interfaces a través de objetos de instancia. Las aplicaciones no deben instanciar las clases en sí mismas; deben usar las funciones de creación disponibles en el objeto Document. Las interfaces derivadas admiten todas las operaciones (y atributos) de las interfaces base, además de cualquier operación nueva.
- Las operaciones se utilizan como métodos. Dado que el DOM usa solo parámetros in, los argumentos se pasan en el orden normal (de izquierda a derecha). No hay argumentos opcionales. Las operaciones void retornan None.
- Los atributos IDL se asignan a atributos de instancia. Por compatibilidad con el mapeo del lenguaje OMG IDL para Python, también se puede acceder a un atributo foo a través de los métodos de acceso _get_foo() y _set_foo(). Los atributos readonly no deben modificarse; esto no se aplica en tiempo de ejecución.
- Los tipos short int, unsigned int, unsigned long long y boolean se asignan todos a objetos enteros de Python.
- El tipo DOMString se asigna a cadenas de caracteres de Python. El módulo xml.dom.minidom admite bytes o cadenas de caracteres, pero normalmente producirá cadenas de caracteres. Los valores de tipo DOMString también pueden ser None cuando la especificación DOM del W3C permite tener el valor IDL null.
- Las declaraciones const se asignan a variables en su ámbito respectivo (por ejemplo, xml.dom.minidom.Node.PROCESSING_INSTRUCTION_NODE); no deben modificarse.
- DOMException no está actualmente soportado por el módulo xml.dom.minidom. En su lugar, xml.dom.minidom usa excepciones estándar de Python como TypeError y AttributeError.

- Los objetos de la clase NodeList se implementan usando el tipo lista incorporado de Python. Estos objetos proporcionan la interfaz definida en la especificación DOM, pero en versiones anteriores de Python no son compatibles con la API oficial. Sin embargo, son mucho más «pythónicas» que la interfaz definida en las recomendaciones del W3C.

Ejemplos:

```
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)
```

```
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>')
```

```
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)
```

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

Si desea analizar una cadena de XML, lo que necesita la función "parseString".

xmlString = "<parentNode> <nodo hijo /> <nodo hijo /> <differentchildnode> Un nodo diferente </ differentchildnode> </ parentNode>"
xmlDocument = xml.dom.minidom.parseString (xmlString);

```
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))
```

XPATH

Módulo implementa una API simple y eficiente para analizar y crear datos XML. *Modificado en la versión 3.3:* este módulo utilizará una implementación rápida siempre que esté disponible. En xml.etree.cElementTreedesuso *desde la versión 3.3:* el módulo está en desuso.

Ejemplos:

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

```
import xml.etree.ElementTree as ET

root = ET.fromstring(countrydata)

# Top-level elements
root.findall(".")

# All 'neighbor' grand-children of 'country' children of the top-level
# elements
root.findall("./country/neighbor")

# Nodes with name='Singapore' that have a 'year' child
root.findall(".//year/..[@name='Singapore']")

# 'year' nodes that are children of nodes with name='Singapore'
root.findall(".//*[@name='Singapore']/year")

# All 'neighbor' nodes that are the second child of their parent
root.findall(".//neighbor[2]")
```

```
from xml.etree import ElementTree, ElementInclude

tree = ElementTree.parse("document.xml")
root = tree.getroot()

ElementInclude.include(root)
```

```
<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
```

```
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
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='Sing
root.findall(".//*[@name='Singapore']/year")

# todos los nodos 'neighbor'que son el segundo hijo de su pac
root.findall(".//neighbor[2]")
```