Universidad de San Carlos de Guatemala
Introducción a la Programación de Computadoras 2
Primer Semestre 2021
Bárbara Raquel García Estrada
201903859

Actividad #2

DOM xml y xPath XML para python

- **Uso de DOM para Python**

Modelo de Objetos del Documento (DOM), es un lenguaje API de W3C, este posee acceso y puede modificar 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 implementan interfaces conocidas. El DOM es útil para aplicaciones de acceso directo.

Algunas aplicaciones son imposibles en un modelo orientado a eventos sin acceso a un árbol. Se puede 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.

Las aplicaciones DOM suelen comenzar analizando algún XML en un DOM. Con xml.dom.minidom, esto se hace a través de las funciones de análisis sintáctico:

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

```
dom1 = parse('c:\\temp\\mydata.xml')  # parseo de un archivo XML
datasource = open('c:\\temp\\mydata.xml')
dom2 = parse(datasource)  # parseo de un archivo abierto
dom3 = parseString('<myxml>Some data<empty/> some more data</myxml>')
```

2. 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)))
```

3. Se utiliza el alto y ancho de las imágenes de la variación de dimensiones.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>width/height example</title>
<script>
function init() {
  var arrImages = new Array(3);

  arrImages[0] = document.getElementById("image1");
  arrImages[1] = document.getElementById("image2");
  arrImages[2] = document.getElementById("image3");

  var objOutput = document.getElementById("output");
  var strHtml = "<ul>";

  for (var i = 0; i < arrImages.length; i++) {
    strHtml += "<li>image" + (i+1) +
```

```
            ": height=" + arrImages[i].height +
            ", width=" + arrImages[i].width +
            ", style.height=" + arrImages[i].style.height +
            ", style.width=" + arrImages[i].style.width +
            "<\/li>";
  }

  strHtml += "<\/ul>";

  objOutput.innerHTML = strHtml;
}
</script>
</head>
<body onload="init();">

<p>Image 1: no height, width, or style
  <img id="image1" src="http://www.mozilla.org/images/mozilla-banner.gif">
</p>

<p>Image 2: height="50", width="500", but no style
  <img id="image2"
       src="http://www.mozilla.org/images/mozilla-banner.gif"
       height="50" width="500">
</p>

<p>Image 3: no height, width, but style="height: 50px; width: 500px;"
  <img id="image3"
       src="http://www.mozilla.org/images/mozilla-banner.gif"
       style="height: 50px; width: 500px;">
</p>

<div id="output"> </div>
</body>
</html>
```

4. Al tener un documento DOM, es posible acceder a la estructura XML.

```
dom3 = parseString("<myxml>Some data</myxml>")
assert dom3.documentElement.tagName == "myxml"
```

5. import xml.dom.minidom

```
document = """\
<slideshow>
<title>Demo slideshow</title>
<slide><title>Slide title</title>
<point>This is a demo</point>
<point>Of a program for processing slides</point>
</slide>

<slide><title>Another demo slide</title>
<point>It is important</point>
<point>To have more than</point>
<point>one slide</point>
</slide>
</slideshow>
"""

dom = xml.dom.minidom.parseString(document)

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

def handleSlideshow(slideshow):
    print("<html>")
   handleSlideshowTitle(slideshow.getElementsByTagName("title"[0])
    slides = slideshow.getElementsByTagName("slide")
    handleToc(slides)
    handleSlides(slides)
    print("</html>")

def handleSlides(slides):
    for slide in slides:
        handleSlide(slide)

def handleSlide(slide):
```

```
    handleSlideTitle(slide.getElementsByTagName("title")[0])
    handlePoints(slide.getElementsByTagName("point"))

def handleSlideshowTitle(title):
    print("<title>%s</title>" % getText(title.childNodes))

def handleSlideTitle(title):
    print("<h2>%s</h2>" % getText(title.childNodes))

def handlePoints(points):
    print("<ul>")
    for point in points:
        handlePoint(point)
    print("</ul>")

def handlePoint(point):
    print("<li>%s</li>" % getText(point.childNodes))

def handleToc(slides):
    for slide in slides:
        title = slide.getElementsByTagName("title")[0]
        print("<p>%s</p>" % getText(title.childNodes))

handleSlideshow(dom)
```

6.
```
from xml.dom import minidom

doc = minidom.parse("staff.xml")

# doc.getElementsByTagName retorna 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))
```

- **Uso de xPath XML para Python**

XPath es un componente principal y básico del estándar XSLT. XPath se puede utilizar para recorrer los elementos, atributos, texto, instrucciones de procesamiento, comentarios, espacios de nombres y documentos en un documento de Lenguaje de marcado extensible (XML). Es una recomendación del W3C que contiene una biblioteca con más de 200 funciones integradas. XPath es la sintaxis para definir partes de un documento XML.

XSLT es el lenguaje de hoja de estilo para archivos XML. Con XSLT puede transformar documentos XML en otros formatos, como XHTML. XQuery se trata de consultar datos XML. XQuery está diseñado para consultar cualquier cosa que pueda aparecer como XML, incluidas las bases de datos. La vinculación en XML se divide en dos partes: XLink y XPointer. XLink y XPointer definen una forma estándar de crear hipervínculos en documentos XML.

XPath aborda una parte específica del documento. Modela un documento XML como un árbol de nodos. Una expresión de XPath es una técnica para navegar y seleccionar nodos del documento.

1. Se podrá utilizar un archivo xml:

```
<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>
```

Y buscar ‘Books’:

```
import xml.etree.cElementTree as ET
tree = ET.parse('sample.xml')
tree.findall('Books/Book')
```

2. Para buscar un libro con el título de 'The Colour of Magic':

```
import xml.etree.cElementTree as ET
tree = ET.parse('sample.xml')
tree.find("Books/Book[Title='The Colour of Magic']")
```

3. 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()

4. #parse the document into a DOM tree rdf_tree = xml.dom.minidom.parse("install.rdf") #read the default namespace and prefix from the root node context = xpath.XPathContext(rdf_tree) name = context.findvalue("//em:id", rdf_tree) version = context.findvalue("//em:version", rdf_tree) #<Description/> inherits the default RDF namespace resource_nodes = context.find("//Description/following-sibling::*", rdf_tree)

5. def xpath_ns(tree, expr): "Parse a simple expression and prepend namespace wildcards where unspecified." qual = lambda n: n if not n or ':' in n else '*[local-name() = "%s"]' % n expr = '/'.join(qual(n) for n in expr.split('/')) nsmap = dict((k, v) for k, v in tree.nsmap.items() if k) return tree.xpath(expr, namespaces=nsmap) doc = '''<root xmlns="http://really-long-namespace.uri" xmlns:other="http://with-ambivalent.end/#"> <other:elem/> </root>''' tree = lxml.etree.fromstring(doc) print xpath_ns(tree, '/root') print xpath_ns(tree, '/root/elem') print xpath_ns(tree, '/root/other:elem')

6. def post(self, html):
        """

```
    Try to play with request ...
    """

    import urllib2

    response = urllib2.urlopen('file://%s' % html)
    data = response.read()

    post = etree.HTML(data)

    # find text function

    find_text = etree.XPath("//text()", smart_strings=False)

    LOG.info(find_text(post))

    post.clear()
```

7.
```
def test_parse_rule():
    """Ensure parse_rule returns expected output."""
    expr = XPath("//Num")
    assert parse_rule(
        rule_name='',
        rule_values=dict(
            description='',
            expr=expr,
            example="a = 1",
            instead="a = int('1')",
            settings=Settings(included=[], excluded=[], allow_ignore=True),
        )
    ) == Rule(
        name='',
        description='',
        expr=expr,
        example="a = 1",
        instead="a = int('1')",
        settings=Settings(included=[], excluded=[], allow_ignore=True)
    )
```