Parsear XML

Antonio Espín Herranz

Contenidos

- Parsear documentos XML:
  - Con DOM
  - Con SAX
  - Crear documentos XML y grabarlos en un fichero.

xml

- Python 3 proporciona el **módulo xml** para parsear documentos xml.

- El módulo se suministra con la instalación estándar y no es necesario instalar ninguna librería adicional.

- Existen dos formas de parsear un documento xml:
  - SAX
  - DOM

SAX vs DOM

- No carga el documento en memoria.
- Recorre el documento generando eventos según va leyendo apertura / cierre de etiquetas, texto, etc.
- Más rápido pero sin acceso a los nodos una vez que se han leído.
- Consume menos memoria.

- Carga toda la estructura del documento en memoria, mediante una jerarquía de objetos anidados.
- Consume más memoria y es un poco más lento.
- Aporta la estructura de nodos en memoria y es accesible mediante métodos.

xml

- El módulo más habitual para xml es: **xml.etree** (ElementTree).

- Hay otros módulos llamados:
  - xml.dom
  - xml.dom.minidom
  - xml.dom.pulldom
  - xml.sax

DOM: ElementTree

- Para **analizar** el documento, indicar el fichero en la instrucción **with**.

- Después llamar al método **parse**.

- Si va todo bien lo cargará en memoria e imprime una referencia al objeto:

```
from xml.etree import ElementTree
with open('fichero.xml','rt') as f:
    tree = ElementTree.parse(f)
print(tree)
```

Element

- Disponemos de un **iterador** para recorrer los nodos del árbol xml.

- Cada nodo se corresponde con un objeto **Element**.

- Para imprimir las etiquetas de todos los nodos:

```
for nodo in tree.iter():
    print(nodo.tag)
```

Métodos

- Obtener el **nodo raíz** (de tipo Element):

  root = tree.getroot()

- Obtener una **lista de hijos** del nodo raíz. Lista de objetos Element:

  root.getchildren()

  for nodo in root.getchildren():
      print('\t',nodo, nodo.tag, nodo.text, nodo.attrib)

  - nodo: imprimirá una referencia al objeto.
  - nodo.**tag**: la etiqueta del nodo.
  - nodo.**text**: el texto del nodo, si no tiene, será None.
  - nodo.**attrib**: Los atributos del nodo. Es un dict., con get(clave) podemos obtener el valor.

Buscar nodos

- Podemos utilizar lenguaje Xpath (lenguaje de consulta para un fichero XML), para localizar nodos:
- A partir del nodo raíz buscamos todos los nodo titulo:
  - Con la expresión: **.//titulo**
    - Busca nodos a cualquier nivel.

```
for t in root.findall('.//titulo'):
    print('\tTitulo: ', t.text)
```

    - Para los apellidos del autor:
      .//autor/apellido

Atributos de un nodo

```
<?xml version="1.0" encoding="UTF-8"?>
<top>
  <child>Regular text.</child>
  <child_with_tail>Regular text.</child_with_tail>"Tail" text.
  <with_attributes name="value" foo="bar" />
  <entity_expansion attribute="This &#38; That">
    That &#38; This
  </entity_expansion>
</top>
```

```
from xml.etree import ElementTree
with open('data.xml', 'rt') as f:
    tree = ElementTree.parse(f)
    node = tree.find('./with_attributes')
    print(node.tag)
    for name, value in sorted(node.attrib.items()):
        print(' %-4s = "%s"' % (name, value))
```

SAX: Eventos

- Los datos se extraen del documento a la vez que se itera.
- Los eventos se producen cuando se abre una etiqueta, se cierra, etc.
- No se carga todo el documento en memoria.
- Si queremos acceder a la información una vez procesado el documento tenemos que mantener la información en algún tipo de estructura.

Tipos de eventos

- **start**
  - Cuando se encuentra una nueva etiqueta.
- **end**
  - Cuando se cierra una etiqueta.
- **start-ns**
  - Se inicia la declaración de un namespace.
- **end-ns**
  - Finaliza la declaración de un namespace.
- **iterparse**()
  - Devuelve un iterable que produce tuplas que contienen el nombre del evento y el nodo que lo desencadena.

Análisis de Eventos

```
from xml.etree.ElementTree import iterparse
EVENT_NAMES = ['start', 'end', 'start-ns', 'end-ns']
for (event, node) in iterparse(fichero, EVENT_NAMES):
    print(event, node)
```

```
start <Element 'bib' at 0x02025930>
start <Element 'libro' at 0x02025990>
start <Element 'titulo' at 0x020259F0>
end <Element 'titulo' at 0x020259F0>
start <Element 'autor' at 0x02025A50>
start <Element 'apellido' at 0x02025AB0>
end <Element 'apellido' at 0x02025AB0>
start <Element 'nombre' at 0x02025AE0>
end <Element 'nombre' at 0x02025AE0>
end <Element 'autor' at 0x02025A50>
start <Element 'editorial' at 0x02025B10>
end <Element 'editorial' at 0x02025B10>
start <Element 'precio' at 0x02025B40>
```

Construir nodos en XML

- La clase xml.etree.ElementTree a parte de analizar documentos XML también permite construir documentos XML a partir de objetos Element.

- Una vez creada la estructura en forma de árbol se puede grabar en un fichero.

Tipos de nodos

- **Element**()
  - Crea un nodo estándar.
- **SubElement**()
  - Añade un nuevo nodo al padre.
- **Comment**()
  - Crea un nodo de comentario.
- **tostring**()
  - Convierte el nodo y todos sus descendientes en texto.

- Para importar:

  from xml.etree.ElementTree import **Element**, **SubElement**, **Comment**, **tostring**

Ejemplo

- **# Crear el nodo raíz:**
- top = Element('raiz')
- **# Configurar un atributo del nodo:**
- top.set('version', '1.0')
- **# Crear un nodo comentario:**
- comment = Comment('Ejemplo de comentario')
- **# Añadir el nodo comentario a la raíz:**
- top.append(comment)
- **# Añadir un elemento hijo:**
- hijo = SubElement(top, 'hijo1')
- hijo.text = 'hijo1_txt‘
- print(tostring(top))

```
b'<raiz version="1.0"><!--un comentario--><hijo1>hijo1_txt</hijo1></raiz>'
Presione una tecla para continuar . . .
```

Serializando XML a un Stream

- Grabar el XML en un fichero:

```
from xml.etree import ElementTree
tree = ElementTree.ElementTree()
tree._setroot(top)
tree.write('ejemplo.xml')
```

- También se pueden generar nodos de la siguiente forma:

```
hijos = [Element('nodo', num=str(i)) for i in range(5)]
top.extend(hijos)
```

```
b'<raiz version="1.0"><nodo num="0" /><nodo num="1" /><nodo num="2" /><nodo num=
"3" /><nodo num="4" /></raiz>'
Presione una tecla para continuar . . .
```

Enlaces

- https://pymotw.com/3/xml.etree.ElementTree/parse.html
- https://docs.python.org/3/library/xml.etree.elementtree.html