

Procesamiento de archivos XML en Python

Juan Alvarado, Estefania Lozano & Henry Lasso

February 19, 2014

1 Introducción

El propósito de nuestro proyecto es que mediante un analizador de texto en este caso el parseo comprobar si un archivo xml esta bien formado ,comprobar si el parseo es válido y poder realizar consultas sobre la información contenida en el archivo .

El lenguaje de programación que utilizaremos es Python que cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos.

La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas

Python permite escribir programas compactos y legibles. Los programas en Python son típicamente más cortos que sus programas equivalentes en C, C++ o Java por varios motivos:

- los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción
- la agrupación de instrucciones se hace por sangría en vez de llaves de apertura y cierre
- no es necesario declarar variables ni argumentos.

Estas son suficientes razones por las cuales Python es un lenguaje adecuado para hacer un parse de XML a una estructura manejable.

2 Alcance

Consiste en extraer la información que contiene un archivo xml, analizar su contenido , representarlo en una estructura y realizar funciones que nos permitan consultar su contenido.

3 Descripción

Se podría pensar en un documento XML como una estructura recursiva. Por tanto lo que buscamos es poder comprender en un solo vistazo las funcionalidades y/o módulos enteros. Con python pretendemos definir recursividad que implemente las estructuras anidadas del documento XML. Ya que en general, en cualquier XML se encuentran estructuras definidas y etiquetadas. Es decir las definidas con los símbolos de mayor o menor: <f>abc</f>. Que pueden ser en parejas (<f> ,</f>) o únicas (apertura y cierre) (<f ... />). Y que recursivamente pueden contener más parejas de etiquetas o texto plano. Nuestra decisión de programación fue leer el documento XML y su conjunto de datos, para representarlo como una Lista en Python. Pero no una Lista de Enteros o de Cadenas de Texto sino una Lista de un Tipo de Dato.

Haskell permite que definamos un Dato con sus características y a su vez el Tipo de Dato de las mismas. Por lo que definimos tres de estas en nuestro proyecto: Device, Group y Capability. Y al final lo que tuvimos como resultado fue una Lista que almacena Device.

- Lo que tenemos es una Lista de Devices que tiene como elementos idD , useragent, actualdevice , fallback y una lista de groups. Por medio del fallback del Device se hace uso de la orientación a objetos de Python y se tiene un puntero al Device que es fallback.
- La lista de Group tiene elementos de tipo Group con las siguientes características : idG y una lista de capabilities .
- La lista de Capabilities tiene elementos de tipo Capability con estas características : name y value .

```
class Capability:
    def __init__(self, name=None, value=None):
        self.name = name
        self.value = value

class Group:
    def __init__(self, id=None, capabilities=None):
        self.id = id
        if capabilities is None:
            self.capabilities = []
        else:
            self.capabilities = capabilities

class Device:
    def __init__(self, id=None, user_agent=None,
        actual_device_root=None, fall_back=None, groups=None):
        self.id=id
        self.user_agent=user_agent
        self.actual_device_root=actual_device_root
        self.fall_back=fall_back
        if groups is None:
            self.groups = []
        else:
            self.groups = groups
```

Los tipos de datos construidos en Python

4 ¿Qué se implementó y qué no se implementó?

4.1 Lo implementado

Una vez extraída la información del archivo xml, la representamos en una lista de Devices que contiene elementos de tipo Device que se puede enlazar a su device fallback, contiene una lista de Groups que tienen elementos de tipo Group y cada Group contiene una lista de Capabilities que tienen elementos de tipo Capability.

La lista creada en Python de Devices

```
>>> ===== RESTART =====
>>> from test import *
>>> f=open("cml.xml")
>>> list=listarDevices(f)
>>> len(list)
15438
```

Un ejemplo de lo que hay en la lista obtenida, y sus respectivos valores

```
>>> list[0].id
'generic'
>>> list[0].groups[0].id
'product_info'
>>> list[0].groups[5].capabilities[0].name
'html_web_3_2'
```

Con la lista de Devices ya obtenida podemos consultar la información de cada Device , Group o Capability.

Consultas en Python

```
>>> lista2=findCapDevices(list,"can_assign_phone_number","true")
>>> len(lista2)
15025
>>> lista2[0].id
'generic'
>>> lista2[5].id
'nokia_generic_series30'
```

4.2 Lo no implementado

5 Observaciones


- Toda la sintaxis de Python esta basada en el correcto acomodo de espacios e indentación, lo cual se presta a confusiones. En este caso es recomendable encontrar un buen editor de texto que nos marque correctamente los espacios, o nos ayude con colores, figuras o marcas dependiendo del lenguaje en el que trabajemos. Aunque en este caso Python. El editor de texto usado fue Sublime Text 2

Correcta indentación en Python con Sublime Text 2

```

78 while not re.search("<devices>", a) :
79     a=next(i)
80 while not re.search("</devices>", a) :
81     m=re.search("<device\s", a)
82     if m:
83         dev=leerDevDeString(a)
84         if not re.search("/>", a) :
85             while not re.search("</device>", a) :
86                 gm=re.search("<group\s", a)
87                 if gm:
88                     tempgroup=leerGroupDeString(a)
89                     while not re.search("</group>", a) :
90                         cap=leerCapDeString(a)
91                         if cap:
92                             tempgroup.capabilities.append(cap)
93                         a=next(i)
94                     dev.groups.append(tempgroup)
95                 a=next(i)
96             devlist.append(dev)
97     a=next(i)
98

```



- Para leer el archivo XML, y hacer el respectivo linkeo al fallback de cada dispositivo tuvimos la necesidad de hacer una búsqueda, y lo hacíamos de manera lineal. En este caso fue necesario encontrar un algoritmo de Búsqueda eficiente, para mejorar el tiempo en el que se cargaba la lista de Devices en nuestra estructura.

6 Conclusiones

- Python es un lenguaje muy cómodo y fácil de implementar. El hecho de que no se necesiten llaves, ni corchetes para las funciones, que no se definan los tipos de datos de las variables, y que se puedan crear de forma muy sencilla; hace a Python un lenguaje muy versátil y elegible como favorito a la hora de programar.
- Python permite separar nuestro programa en módulos que pueden reusarse en otros programas en Python.
- En cualquier búsqueda que realizemos es muy importante analizar el algoritmos que usemos, no es necesario que nosotros creemos uno. Lo importante es saber que existen ya implementados y que están disponibles para nuestro uso. Siempre en busca de una programación eficiente.