Blog de Norore

Bioinformaticienne passionnée de biologie, de programmation, de libre et de jeux-vidéo

Parser un fichier XML avec espaces de noms à coup de Python

Rédigé par Norore 12 juin 2017 2 commentaires

Récemment un ami a solicité mon aide dans le cadre d'un de ses projets personnels. Manquant de compétences et de temps pour les acquérir, il m'a alors demandé si je pouvais développer un script pour convertir un fichier au format XML vers le format JSON. Sur le papier, c'est très simple et facile à faire. Dans la pratique, il s'est avéré que le découpage simple du XML n'était pas possible car des espaces de noms étaient présents dans chaque balise. J'ai donc dû trouver comment faire, aussi je viens vers vous pour vous expliquer comment le refaire.

Je vous ai perdu dans cette courte introduction ? Pas de panique, on va y aller en douceur. :-)

Le XML ? Késako?

Un peu d'histoire

En 1969, Charles Goldfarb, alors chef de projet chez IBM, conçoit le GML, qui le fera mondialement connaître. C'est fort de ce succès qu'il quitte son emploi et créé le SGML, en 1986, en tant que norme ISO. Pourquoi je vous parle de cela ? Tout simplement parce que le XML (1996), que l'on présente souvent comme un héritier du HTML, définit en 1989 par Tim Berners-Lee, découle en fait du SGML, puisque son papa en est issu.

Nous avons donc, dans l'ordre chronologique :

1. 1969, GML
2. 1986, SGML
3. 1989, HTML
4. 1996, XML

Les langages de balisages peuvent donc être considérés comme robustes et éprouvés par le temps et les normes ISO auxquelles ils appartiennent.

Le gros avantage de ces langages réside dans le fait que la personne qui l'utilise peut définir ses propres balises, ce qui s'avère intéressant pour nombre de projets de programmation ou de projets scientifiques, notamment si l'on cherche à obtenir un format robuste et pérenne.

Syntaxe de base

La syntaxe de base est la même que celle du HTML. À un ou deux détails près. Ainsi que je vous l'ai indiqué dans le point précédent, le but premier du XML est d'avoir un document structuré et qui respecte une logique de hiérarchie. Aussi c'est au concepteur du fichier de définir la structure exacte du document et quels seront les noms, ou tags, de ses balises.

Prenons l'exemple XML suivant, fournit par le w3schools, et décortiquons-le à la main :

```
<menu_ptit_dej>
  <plat>
    <nom>Gaufres belges</nom>
    <prix>5,95e</prix>
    <description>Deux de nos fameuses gaufres belges recouvertes de véritable sirop d'é
    <calories>650</calories>
  </plat>
  <plat>
    <nom>Gaufres belges aux fraises</nom>
    <prix>7,95e</prix>
    <description>Gaufres belges allégées couvertes de fraises et de crème fouettée</des
    <calories>900</calories>
  </plat>
  <plat>
    <nom>Gaufres belges aux fruits des bois</nom>
    <prix>8,95e</prix>
    <description>Gaufres belges allégées couvertes de fruits des bois et de crème fouet
    <calories>900</calories>
  </plat>
  <plat>
    <nom>Tartines françaises</nom>
    <prix>4,50e</prix>
    <description>Tranches épaisses de notre pain au levain maison</description>
    <calories>600</calories>
  </plat>
  <plat>
    <nom>Petit déjeuner façon maison</nom>
    <prix>6,95e</prix>
    <description>Deux œufs, bacon ou saucisse, tartine et notre célèbre crème de marron
```

```
    <calories>950</calories>
  </plat>
</menu_ptit_dej>
```

Il faut tout d'abord voir un fichier XML comme un arbre, avec une racine, des branches et des feuilles.

Nous avons donc la racine `menu_ptit_dej` qui contient 4 branches `plat`. Chaque branche contient 4 feuilles :

- `nom`
- `prix`
- `description`
- `calories`

L'ordre des branches et des feuilles n'a pas d'importance, seule la hiérarchie l'est !

Prenons l'exemple du troisième plat. Ce plat est intitulé "Tartines françaises" (feuille `nom`), il s'agit de "Tranches épaisses de pain au levain maison" (feuille `description`), il coûte 4€50 (feuille `prix`) et a un apport calorique de 600 (feuille `calories`).

Ok, et les espaces de noms dans tout ça ?

Les espaces de noms sont là pour lever une ambiguïté possible au niveau des balises XML. Si l'on prend l'exemple suivant, toujours fourni par le W3C Schools, dans lequel nous définissons un tableau en HTML (`table`) et un meuble, au hasard une table. Dans ce cas là, vous ne pourrez pas savoir qui est quoi !

```
<table>
  <tr>
    <td>Pommes</td>
    <td>Bananes</td>
  </tr>
</table>

<table>
  <nom>Table basse africaine</nom>
  <largeur>80</largeur>
  <longueur>120</longueur>
</table>
```

Oui, bon, lorsqu'on lit le fichier, on peut en déduire que l'on a, dans un cas, un tableau qui liste des fruits, et dans l'autre, une table basse. Mais comment dire à son programme que, dans un cas, c'est un tableau HTML, dans un autre, un meuble ? Grâce aux espaces de noms !

Les espaces de noms permettent de préfixer nos balises XML et d'indiquer à votre moteur de découpage quel action y appliquer. Reprenons nos tables et précisons avec `h` les éléments HTML, et avec `m` les éléments de mobiliers :

```
<racine
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:m="https://www.w3schools.com/meuble">

<h:table>
  <h:tr>
    <h:td>Pommes</h:td>
    <h:td>Bananes</h:td>
  </h:tr>
```

```
</h:table>

<m:table>
  <m:nom>Table basse africaine</m:nom>
  <m:largeur>80</m:largeur>
  <m:longueur>120</m:longueur>
</m:table>

</racine>
```

Nous voici donc avec un important changement au niveau du code, mais qu'est-ce que l'on a bien pu faire ? Décortiquons les nouveautés.

- la balise `root` permet d'englober l'ensemble des balises XML. Rien de très compliqué. Les attributs `xmlns` permettent de définir à quoi correspondent les espaces de noms (`ns`) des balises XML.
- les balises commençant par `h:` indiquent que celles-ci correspondent à l'espace de nom `h` défini par l'adresse web `http://www.w3.org/TR/html4/`, définissant, ici, un objet HTML
- les balises commençant par `m:` indiquent que celles-ci correspondent à l'espace de nom `m` défini par l'adresse web `https://www.w3schools.com/meuble`, définissant, ici, un objet meuble

À noter que les adresses indiquées dans les attributs `xmlns` ne sont pas destinées à être utilisées pour récupérer des informations. Cependant, elles peuvent être utiles pour pointer sur une documentation, par exemple, pour aider les utilisateurs du fichier à savoir quelles sont les balises utilisées par votre outil. N'essayez donc pas de suivre les liens dans ces exemples, vous aurez juste une belle erreur 404 ;-) !

Voyons maintenant ce que nous pouvons faire avec ces deux formats à l'aide de ~~notre~~ mon langage de programmation favori : Python !

Parser un fichier XML simple

Avant de pouvoir extraire les données d'un fichier XML, vous devez vous assurer que vous disposez du module Python lxml, vous pouvez l'installer facilement à l'aide de la commande pip :

```
pip install lxml
```

Voyons maintenant comment utiliser la bibliothèque pour découper notre fichier XML. Commençons par le commencement :

```
#!/usr/bin/env python

from lxml import etree

fichier = "exemple1.xml"
arbre = etree.parse(fichier)
racine = arbre.getroot()
```

À partir de la bibliothèque lxml, nous importons la classe `etree`. Ensuite, nous créons une variable `fichier` qui contient le nom du fichier XML à analyser. Nous initialisons la classe `etree` à l'aide de la méthode `parse` sur la variable `fichier`, afin d'avoir un découpage sous forme d'arbre de notre document. Puis, on se place au niveau de la racine de l'arbre avec la méthode `getroot`.

À partir de là, il existe quatre façons différentes de parcourir notre arbre. Vous pourrez, voire devrez, adapter votre script en fonction de votre fichier XML.

Méthode naïve

Je qualifie cette méthode de naïve car elle consiste, pour chaque premier nœud de l'arbre, de simplement récupérer les informations dans les feuilles (c'est un arbre pour ceux qui ne suivent plus !), sans chercher à établir un affichage hiérarchique :

```
for noeud in arbre.xpath('//menu_ptit_dej'):
    for nom in noeud.xpath('plat/nom'):
        print(nom.text)
    for prix in noeud.xpath('plat/prix'):
        print(prix.text)
    for calories in noeud.xpath('plat/calories'):
        print(calories.text)
    for description in noeud.xpath('plat/description'):
        print(description.text)
```

La fonction `text` assignée à chaque feuille parcourue (`nom`, `prix`, `calories` et `description`) permet de récupérer le texte associé, définit dans l'attribut XML fournit par le chemin `xpath`.

Le résultat affiché est le suivant :

```
Nom : Gaufres belges
Nom : Gaufres belges aux fraises
Nom : Gaufres belges aux fruits des bois
Nom : Tartines françaises
Nom : Petit déjeuner façon maison
Prix : 5,95e
Prix : 7,95e
Prix : 8,95e
Prix : 4,50e
Prix : 6,95e
Calories : 650
Calories : 900
Calories : 900
Calories : 600
Calories : 950
Description : Deux de nos fameuses gaufres belges recouvertes de véritable sirop d’érab
Description : Gaufres belges allégées couvertes de fraises et de crème fouettée
Description : Gaufres belges allégées couvertes de fruits des bois et de crème fouettée
Description : Tranches épaisses de notre pain au levain maison
Description : Deux œufs, bacon ou saucisse, tartine et notre célèbre crème de marrons
```

Le résultat est assez moche, tout est récupéré par grappe, sans respect de la hiérarchie, mais on peut au moins récupérer nos données. Mais peut-on faire mieux ? Testons une autre solution, un peu plus sioux !

Méthode Tipiak

Cette méthode, contrairement à la méthode naïve, consiste à jouer avec les chemins `xpath`, les listes Python et la méthode `text`.

```
for noeud in arbre.xpath('//menu_ptit_dej'):
    for plat in noeud.iter('plat'):
        print("Nom : {}".format(plat.xpath("nom")[0].text))
        print("Prix : {}".format(plat.xpath("prix")[0].text))
        print("Calories : {}".format(plat.xpath("calories")[0].text))
        print("Description : {}".format(plat.xpath("description")[0].text))
```

Le résultat affiché est le suivant :

```
Nom : Gaufres belges
Prix : 5,95e
Calories : 650
Description : Deux de nos fameuses gaufres belges recouvertes de véritable sirop d'éral
Nom : Gaufres belges aux fraises
Prix : 7,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fraises et de crème fouettée
Nom : Gaufres belges aux fruits des bois
Prix : 8,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fruits des bois et de crème fouettée
Nom : Tartines françaises
Prix : 4,50e
Calories : 600
Description : Tranches épaisses de notre pain au levain maison
Nom : Petit déjeuner façon maison
Prix : 6,95e
Calories : 950
Description : Deux œufs, bacon ou saucisse, tartine et notre célèbre crème de marrons
```

C'est déjà mieux ! On peut mieux contrôler ce qui est lu et la façon dont c'est affiché. Par contre, il faut être sûr que le fichier XML sera toujours sous cette présentation pour les listes ! Continuons de fouiner…

Méthode crade

Dans cette méthode, nous allons surtout jouer avec `xpath`, mais en l'améliorant, ce qui induit l'ajout d'une boucle supplémentaire, d'où mon choix de qualifier cette méthode de « crade » !

```
for noeud in arbre.xpath('//menu_ptit_dej'):
    for plat in noeud.xpath('plat'):
        for nom in plat.xpath('nom'):
            print("Nom : {}".format(nom.text))
        for prix in plat.xpath('prix'):
            print("Prix : {}".format(prix.text))
        for calories in plat.xpath('calories'):
            print("Calories : {}".format(calories.text))
        for description in plat.xpath('description'):
            print("Description : {}".format(description.text))
```

Le résultat affiché est le même que pour la méthode « Tipiak » !

```
Nom : Gaufres belges
Prix : 5,95e
Calories : 650
Description : Deux de nos fameuses gaufres belges recouvertes de véritable sirop d'éral
Nom : Gaufres belges aux fraises
Prix : 7,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fraises et de crème fouettée
Nom : Gaufres belges aux fruits des bois
Prix : 8,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fruits des bois et de crème fouettée
Nom : Tartines françaises
Prix : 4,50e
Calories : 600
Description : Tranches épaisses de notre pain au levain maison
Nom : Petit déjeuner façon maison
Prix : 6,95e
Calories : 950
```

```
Description : Deux œufs, bacon ou saucisse, tartine et notre célèbre crème de marrons
```

Oui, non, ça donne le même résultat, mais il y a trop de boucles. Il y a sûrement moyen de faire quelque chose de plus optimisé. Réfléchissons…

Méthode Trouve-tout

Dans cette méthode, nous laissons tomber `xpath`, et allons plutôt jouer avec la méthode `findall`. Que fait cette méthode ? Elle cherche et retourne tous les éléments d'une branche. Nous allons donc récupérer, sous la forme d'une liste, les ensembles de branches et de feuilles pour toutes les branches `plat`. Par chance, les plats ne contiennent que des feuilles dans cet exemple ! Pour récupérer les feuilles, il nous suffit d'utiliser la méthode `find`.

```
plats = arbre.findall("plat")
for p in plats:
    print("Nom : {}".format(p.find("nom").text))
    print("Prix : {}".format(p.find("prix").text))
    print("Calories : {}".format(p.find("calories").text))
    print("Description : {}".format(p.find("description").text))
```

Ce qui nous donne le résultat final :

```
Nom : Gaufres belges
Prix : 5,95e
Calories : 650
Description : Deux de nos fameuses gaufres belges recouvertes de véritable sirop d'éral
Nom : Gaufres belges aux fraises
Prix : 7,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fraises et de crème fouettée
Nom : Gaufres belges aux fruits des bois
Prix : 8,95e
Calories : 900
Description : Gaufres belges allégées couvertes de fruits des bois et de crème fouettée
Nom : Tartines françaises
Prix : 4,50e
Calories : 600
Description : Tranches épaisses de notre pain au levain maison
Nom : Petit déjeuner façon maison
Prix : 6,95e
Calories : 950
Description : Deux œufs, bacon ou saucisse, tartine et notre célèbre crème de marrons
```

OK, le résultat est le même que les précédents exemples. Mais cette méthode présente le grand avantage de ne plus utiliser qu'une seule boucle, ce qui, en terme de calcul, peut nous faire gagner du temps d'exécution !

Parser un fichier XML avec espaces de noms

Contrairement aux exemples précédents où nous n'avions joué qu'avec la méthode `parse` simple fournie par `etree`, ici nous allons utiliser, en plus, la méthode `parse` de la classe `ElementTree` fournie par `etree`.

Dans un premier temps, à l'aide de `etree.parse()`, nous allons chercher la liste de tous les espaces de noms dans `<racine>`. Puis, une fois ceci fait, nous initialisons la classe `ElementTree`, à laquelle nous utilisons la méthode `parse` sur notre fichier XML. La syntaxe utilisée ensuite par les méthodes `findall` et `find` sont similaires, au détail près que nous devons leur indiquer un second paramètre : la liste des

espaces de noms, stockée dans la variable `tagmap` !

Voici le code source utilisé pour extraire les donnés du second fichier présenté en première partie, à savoir le fichier avec les tables :

```
#!/usr/bin/env python

from lxml import etree
from lxml.etree import ElementTree

fichier = "exemple2.xml"
arbre = etree.parse(fichier)
racine = arbre.getroot()
tagmap = racine.nsmap

et = ElementTree()
arbre = et.parse(fichier)

listes = arbre.findall("h:table", tagmap)
paniers, produits = [], []
for l in listes:
    paniers.extend(l.findall("h:tr", tagmap))
for p in paniers:
    produits.extend(p.findall("h:td", tagmap))
print("J'ai une liste de {} produits a acheter :".format(len(produits)))
for i, p in enumerate(produits):
    print("  {}. {}".format(i+1, p.text))

meubles = arbre.findall("m:table", tagmap)
print("J'ai {} tables chez moi :".format(len(meubles)))
for m in meubles:
    print("  Nom : {}".format(m.find("m:nom", tagmap).text))
    print("  Longueur : {}".format(m.find("m:longueur", tagmap).text))
    print("  Largeur : {}".format(m.find("m:largeur", tagmap).text))
```

Ce qui nous donne le résultat suivant :

```
J'ai une liste de 2 produits à acheter :
  1. Pommes
  2. Bananes
J'ai 1 tables chez moi :
  Nom : Table basse africaine
  Longueur : 120
  Largeur : 80
```

Et pour les plus curieux d'entre-vous qui auront la flemme de tester ce que contient la variable `tagmap` :

```
In [53]: tagmap
Out[53]: {'h': 'http://www.w3.org/TR/html4/', 'm': 'https://www.w3schools.com/meuble'}
```

Conclusion personnelle

Nous avons donc vu deux outils puissants aujourd'hui : le format de fichier XML et la bibliothèque Python lxml. Ces deux outils, combinés ensemble, peuvent nous permettre de récupérer de façon efficace des informations multiples. J'apprécie beaucoup la combinaison de ces deux outils et, bien que ne maîtrisant pas encore toute la puissance de lxml, j'ai beaucoup aimé devoir relever ce défi ! La documentation de lxml est très complète, bien que j'ai eu du mal à bien saisir toutes les fonctionalités, et il y a sûrement des façons plus simples et rapides pour découper des fichiers XML avec des espaces de

noms.

J'espère que ce billet vous aura plu et qu'il vous aura été utile, n'hésitez pas à passer par les commentaires si vous avez des remarques ou des améliorations de code à fournir pour faciliter l'usage de lxml !

Source de l'image d'accroche : un pile de troncs d'arbres dans une forêt. Photographié par Life-Of-Pix, sous licence CC0 sur Pixabay

Mise à jour le 28 Août 2017 suite à une petite discussion sur le réseau social Mastodon. Merci Stéphane Borzmeyer pour vos suggestions, dont j'ai tenu compte !

Classé dans : Python, Programmation Mots clés : programmation, python, tutoriel

2 commentaires

#1 mercredi 11 octobre 2017 - 22:59 - dedalios a dit :

> Sous windows10 cela ne marche pas
>
> from lxml import etree
> from lxml.etree import ElementTree
>
> Unresolved import etree
> Unresolved import ElementTree

Répondre

#2 jeudi 12 octobre 2017 - 14:57 - Norore a dit :

> Bonjour, avez-vous bien installé lxml ? Je suppose que oui, sinon Python vous l'aurait également signalé. Je ne peux malheureusement pas vous aider pour Windows, n'utilisant moi-même plus ce système d'exploitation. Peut-être trouverez-vous plus d'informations sur le site des développeurs du module ?
> Bonne journée !

Répondre

Écrire un commentaire

Votre nom ou pseudo :
Votre adresse e-mail (facultatif) :
Votre site Internet (facultatif) :

Contenu de votre message :
Vérification anti-spam Quelle est la troisième lettre du mot vabf ?
Envoyer votre commentaire

Fil RSS des commentaires de cet article

Catégories

- Blog (3)
- Web (2)
- Python (1)
- Programmation (4)
- Linux (5)
- Réflexion (3)
- Bioinformatique (2)
- Serveur web (3)
- Logiciel libre (4)

Derniers articles

- Migration de services
- Parser un fichier XML avec espaces de noms à coup de Python
- Quel serait mon emploi de rêve ?
- PluXML, Nginx et PHP 7 sont dans un bateau
- Installer Krita sous Linux Mint

Mots clés

- blog
- bavardage
- documentation
- réflexion
- programmation
- ubuntu
- party
- installation
- événement
- bioinformatique
- introduction
- vulgarisation
- linux
- logiciel libre
- graphisme
- krita
- tutoriel
- nginx
- serveur web
- php-fpm
- python
- travail
- migration de service

Derniers commentaires

- Bonjour, avez-vous bien instal... par Norore
- Sous windows10 cela ne marche ... par dedalios
- Bon article je partage ! par Jean

- Merci, je vais y jeter un œ... par Norore
- Juste un message d'encourageme... par pyg

Archives

- octobre 2017 (1)
- juin 2017 (1)
- mai 2017 (1)
- 2016 (8)

RSS

- Fil des articles
- Fil des commentaires

Blog de Norore - Bioinformaticienne passionnée de biologie, de programmation, de libre et de jeux-vidéo

Généré par PluXml en 0.014s - Administration

- Fil des articles
- Fil des commentaires
- Haut de page