Créer des documents XML depuis un programme

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 9 octobre 2006. Dernière mise à jour le 19 novembre 2008

https://www.bortzmeyer.org/creer-xml-par-programme.html

On trouve sur le Web beaucoup plus de textes expliquant comment analyser du XML que de textes expliquant comment le produire depuis un programme. D'un côté, c'est logique : produire le XML est bien plus facile et ne mérite pas forcément de longues documentations. D'un autre côté, cela pousse certains programmeurs à ne pas faire attention et à écrire des programmes qui finissent par produire du XML mal formé.

La méthode la plus évidente pour produire du XML est d'exploiter le fait que XML est un format basé sur le texte et d'écrire donc simplement (les exemples ici sont en Python et en C mais le raisonnement est valable pour tous les autres langages de programmation) :

```
print "<foo>%i</foo>" % val
```

Cela fonctionne, mais cela présente plusieurs inconvénients :

— Comme on traite le XML comme du texte, il est très facile de produire involontairement du XML mal formé et rien ne permet de le détecter : `print "<foo>%i<foo>" % val` passera inaperçu si on ne vérifie pas le XML produit (ce qui serait de toute façon une très bonne idée).
— Ici, le format d'expansion de `val` est `%i`, ce qui se limite aux nombres. Mais si c'était `%s`, qui permet n'importe quelle chaîne de caractères, on pourrait avoir des problèmes si la chaîne contenait un caractère utile à XML comme le < ou bien un caractère non ASCII qu'il faudrait penser à encoder proprement, en conformité avec la déclaration XML.

Il est donc en général préférable de produire le XML à partir d'une structure de donnés de plus haut niveau que la chaîne de caractères. Une méthode évidente est d'utiliser DOM, ici pour produire du XHTML :

```
from xml.dom.ext.reader import Sax2
from xml.dom.ext import PrettyPrint
from xml import xpath
from xml.dom import getDOMImplementation
...
    blurb = u"<attention aux caractères spéciaux"
    ...
    html_result = getDOMImplementation().createDocument(
        None,
        "html",
        getDOMImplementation().createDocumentType(
           "html",
           "-//W3C//DTD XHTML 1.0 Strict//EN",
           "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"))
    head = html_result.documentElement.appendChild(html_result.createElement('head'))
    title = head.appendChild(html_result.createElement('title'))
    title.appendChild(html_result.createTextNode("My site: %s" % \
                                    (blurb)))
    ...
    PrettyPrint(html_result)
```

On crée un mémoire un arbre DOM et on l'affiche avec `PrettyPrint`. Par construction, il sera bien formé (mais pas forcément valide). Et DOM se chargera de l'échappement des caractères spéciaux XML. Les caractères non-ASCII seront proprement représentés (`PrettyPrint` semble imposer UTF-8 en sortie mais d'autres méthodes DOM comme `toprettyxml()` permettent d'autres représentations <http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/157358>).

DOM n'est pas très pythonien. Cette norme est en effet conçue pour être indépendante du langage de programmation utilisé et certaines constructions sont donc plutôt déroutantes en Python, par exemple, pour créer un simple attribut, ici `href` :

```
            link = html_result.createElement("a")
            url = html_result.createAttribute("href")
            url.nodeValue = "http://www.ietf.org/rfc/rfc%i.txt" % int(rfcnum)
            link.setAttributeNode(url)
            link.appendChild(html_result.createTextNode("RFC %i" % int(rfcnum)))
```

.

ElementTree <http://docs.python.org/lib/module-xml.etree.ElementTree.html> peut donc être préféré. ElementTree, désormais intégré à la bibliothèque standard de Python, permet de manipuler (lire et écrire) du XML de manière plus pythonesque et donc plus natutrelle au programmeur Python. En contrepartie, il n'a rien de standard, contrairement à DOM et il faudra donc tout réapprendre si on passe à un autre langage comme Perl.

Voici un exemple analogue en ElementTree :

```
blurb = u"<attention aux caractères spéciaux"

import elementtree.ElementTree as ET

html = ET.Element("html")
```

```
head = ET.SubElement(html, "head")

title = ET.SubElement(head, "title")
title.text = "Mon site: %s" % blurb

body = ET.SubElement(html, "body")

link = ET.SubElement(body, "a")
link.text = "Internet Engineering Task Force"
link.attrib["href"] = "http://www.ietf.org/"

print ET.tostring(html, encoding="UTF-8")

# elementtree.SimpleXMLWriter is an alternative, if we just want to
# write the XML. See http://effbot.org/zone/xml-writer.htm
```

Il existe bien sûr d'autres méthodes (par exemple, pour Python <http://www.xml.com/pub/a/2003/10/15/py-xml.html>) mais je crois avoir déjà couvert trois possibilités intéressantes.

Et pour C ? Voici un exemple avec la libxml <http://xmlsoft.org/>. On n'utilise pas une seule fois la syntaxe XML, c'est la routine `xmlDocDumpFormatMemory` qui se chargera de produire du XML légal :

```
#include <libxml/parser.h>

int
main(void)
{

  xmlNodePtr root, node;
  xmlDocPtr doc;
  xmlChar *xmlbuff;
  int buffersize;

  /* Create the document. */
  doc = xmlNewDoc(BAD_CAST "1.0");
  root = xmlNewNode(NULL, BAD_CAST "root");

  /* Create some nodes */
  node = xmlNewChild(root, NULL, BAD_CAST "node", NULL);
  node = xmlNewChild(node, NULL, BAD_CAST "inside", NULL);
  node = xmlNewChild(root, NULL, BAD_CAST "othernode", NULL);

  /* Put content in a node: note there are special characters so
     encoding is necessary! */
  xmlNodeSetContent(node,
                xmlEncodeSpecialChars(doc, BAD_CAST "text con&tent and <tag>"));

  xmlDocSetRootElement(doc, root);

  /* Dump the document to a buffer and print it for demonstration purposes. */
  xmlDocDumpFormatMemory(doc, &xmlbuff, &buffersize, 1);
  printf((char *) xmlbuff);

}
```

Compilé avec `gcc -Wall -I/usr/include/libxml2 -c create-xml.c && gcc -lxml2 -o create-xml create-xml.o`, ce programme va afficher :

```
% ./create-xml
<?xml version="1.0"?>
<root>
  <node>
    <inside/>
  </node>
  <othernode>text con&amp;amp;tent and &amp;lt;tag&amp;gt;</othernode>
</root>
```

Un programme C qui produit du XML plus complexe ? Il y a par exemple mon implémentation (en ligne sur `https://www.bortzmeyer.org/files/traceroute-nanog-xml.c`) du RFC 5388 [1].

Pour C, il existe également d'autres bibliothèques comme GenX <`http://www.tbray.org/ongoing/When/200x/2004/02/20/GenxStatus`> ou Mini-XML <`http://www.minixml.org/`>.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5388.txt`