

Générer du HTML avec TAL

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 Juin 2007

<http://www.bortzmeyer.org/generer-html-avec-tal.html>

Un programme comme un CGI qui doit générer des pages HTML a tout intérêt à passer par un langage de gabarits comme TAL.

Du simple CGI à la grosse usine à gaz, tout programme accessible par le Web a besoin de produire du HTML pour l'envoyer au navigateur. Il peut, et cela s'est fait pendant longtemps, fabriquer cet HTML entièrement lui-même à coups de (ici en C) :

```
printf("<p>Il ne faut <em>surtout</em>plus programmer en C.<p>\n");
```

mais cette technique fait courir un risque élevé d'envoyer du HTML invalide ; au milieu des `printf`, on perd facilement de vue le code HTML généré. En outre, il est difficile avec cette technique de garder les différentes pages produites par le programme en parfaite cohérence.

Si le HTML produit est du XHTML, on peut faire mieux en faisant générer le fichier XML par un programme (<http://www.bortzmeyer.org/creer-xml-par-programme.html>) avec une bibliothèque qui fait une grande partie du travail comme `ElementTree` en Python. Mais cela veut dire qu'au lieu de travailler avec un fichier XML relativement lisible par tous, on va travailler avec un source Perl ou Java, ce qui n'est certainement pas un progrès, surtout si le XHTML doit pouvoir être modifié par un non-programmeur.

Une autre approche, et qui fait l'objet de ce court article, est d'utiliser le langage TAL. Sa grande force est qu'un document XML comportant du TAL reste un document XML et peut donc être manipulé par des outils XML. Mieux, TAL n'utilise que des **attributs** XML ce qui limite le risque de conflit avec, par exemple, des éditeurs XHTML qui pourraient s'étonner de rencontrer des éléments XML inconnus.

Pour fabriquer du HTML à partir de TAL, on écrit donc un ou plusieurs fichiers XHTML dont les éléments sont ornés d'attribus TAL, par exemple :

```
<p>Il ne faut <em>surtout</em> plus programmer en <span
class="language" tal:content="language"/>.</p>
```

L'attribut `tal:content` va dire à TAL de mettre dans le corps de l'attribut `` la valeur de la variable `language`. Si celle-ci vaut `PHP`, TAL va produire :

```
<p>Il ne faut <em>surtout</em> plus programmer en <span
class="language">PHP</span>.</p>
```

On le voit, TAL garantit que le XML va rester du XML et qu'il sera valide même après l'expansion des variables.

Quelles sont les différentes possibilités d'attributs TAL ? Les principales :

- `tal:content`, déjà vu, met la valeur de la variable comme contenu de l'élément,
- `tal:replace` **remplace** tout l'élément par la valeur de la variable,
- `tal:repeat` effectue une boucle sur la valeur de la variable, ce qui est très pratique pour les listes et les tableaux,

La valeur d'une variable TAL peut être une donnée ordinaire mais peut aussi être un gabarit TAL, ce qui permet des les emboîter.

Voici un exemple simple, où un programme gère une base de données. Un gabarit TAL sert à toutes les pages :

```
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
[
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <link rel="stylesheet" type="text/css" href="/css/registry.css" media="screen"/>
    <title tal:content="title">The title</title>
  </head>
  <body>
    <h1 tal:content="title">The title</h1>
    <div class="content" tal:content="structure content">The content</div>
    <p><a href="/">Home</a></p>
    <hr class="beforefooter"/>
    <p>This site comes without warranty.</p>
  </body>
</html>
```

Les éléments `<title>` et `<h1>` vont prendre comme contenu la valeur de la variable `title`. L'élément `<div>` de classe `content` va recevoir, lui, un autre gabarit TAL (d'où le mot-clé `structure` dans le `tal:content`).

Une simple page comme la page d'accueil utilisera, pour le contenu, un gabarit comme :

<http://www.bortzmeyer.org/generer-html-avec-tal.html>

```
<div>
<p>This is just an experimental Web site. Do not ask questions about it, it is just to
  experiment.</p>
<p>To get <a href="/getall">all the registry</a>.</p>
</div>
```

qui sera passé comme variable `content`.

Par contre, une page plus complexe, mettons celle qui va afficher toute la base de données, va avoir besoin de boucle :

```
<div>
<h2>All the registry</h2>
<table>
<tr><th>Thing</th><th>Origin</th><th>Creation</th></tr>
<tr tal:repeat="thing allthings">
  <td><a tal:attributes="href thing/value" tal:content="thing/value"/></td>
  <td tal:content="thing/origin"/>
  <td tal:content="thing/created"/>
  <td><a tal:attributes="href string:${thing/value}?delete=yes">Delete it</a></td>
</tr>
</table>
</div>
```

Ce gabarit nous montre des fonctions plus avancées de TAL :

- `tal:repeat` prend deux arguments, la variable de boucle (ici `thing`) et le tableau à parcourir (ici `allthings`).
- La notation avec un `/` qui permet d'accéder aux champs d'une variable (ici, une `thing` a trois champs, `value`, `origin` et `created`).
- `tal:attributes` qui permet de calculer les attributs, ici de l'élément `<a>`,
- Enfin, la notation avec le `${...}` qui permet de séparer ce qui est variable (ici `thing/value`) de ce qui est constant (ici l'option `delete=yes`).

Pour incarner ces gabarits avec des vrais valeurs, utilisons l'implémentation `SimpleTal` (<http://www.owlfish.com/software/simpleTAL/>) en Python. Le programme comportera :

```
from simpletal import simpleTAL, simpleTALES, simpleTALUtils

html_page = open("general.xml")
home_content = open("home.xml")

template = simpleTAL.compileXMLTemplate(html_page)
home_template = simpleTAL.compileXMLTemplate(home_content)
context = simpleTALES.Context()

result = simpleTALUtils.FastStringOutput()

context.addGlobal("title", "My home page")
context.addGlobal("content", home_template)
template.expand(context, result)
print result.getvalue()
```

et l'exécution de ce programme produira la page d'accueil en HTML.

Pour le tableau, le programme est presque le même :

<http://www.bortzmeyer.org/generer-html-avec-tal.html>

```
from simpletal import simpleTAL, simpleTALES, simpleTALUtils
import time

html_page = open("general.xml")
list_content = open("list.xml")

template = simpleTAL.compileXMLTemplate(html_page)
list_template = simpleTAL.compileXMLTemplate(list_content)
context = simpleTALES.Context()

# Extract from the database. Here, we simulate:
db_content = [
    {'value': "foo", 'origin': "France", 'created': time.time()},
    {'value': "bar", 'origin': "Brazil", 'created': time.time()},
    {'value': "baz", 'origin': "Turkey", 'created': time.time()},
]
context.addGlobal("allthings", db_content)

result = simpleTALUtils.FastStringOutput()

context.addGlobal("title", "My database")
context.addGlobal("content", list_template)
template.expand(context, result)
print result.getvalue()
```

Une version complète de ce dernier programme est publiée en <http://www.bortzmeyer.org/rest-sql-unicode-exemple.html>.