alsacréations

Niveau

XML en quelques mots

ARTICLE **formats**

Publié par Gilles le 19 Décembre 2007, mis à jour le 19 Mars 2013 (67879 lectures)

xml xsl

Ce tutoriel expose les bases de ce qu'il est nécessaire de connaître quand on doit aborder un document XML. Nous allons commencer par décrire en quoi consiste le format, continuer en décrivant quelles sont les briques qui constituent un document XML, voir quels sont les formats qui permettent de définir de nouveaux langages XML et enfin décrire le principe de la transformation d'XML, un processus qui est au cœur de la portabilité de ce format.

Sommaire du document

Qu'est-ce qu'XML ?

XML signifie eXtensible Markup Language : en français, c'est un *langage de balisage extensible*.

C'est un *langage* :

Cela signifie que ce format de fichier est conçu pour transmettre des informations. À ce titre, il ne faut pas se limiter aux échanges d'informations entre humains, ou entre une application et un humain : XML est tout aussi bien conçu, si ce n'est mieux, pour des échanges entre applications informatiques.

C'est un langage *de balisage* :

Cela signifie qu'on accole aux données des « étiquettes » qui qualifient leur contenu. Par exemple, dans Émile Zola, on étiquettera « Émile » comme étant un prénom, et « Zola » comme étant un nom de famille.

C'est un langage *extensible* :

Cela signifie deux choses en fait :

- selon ses besoins, on peut définir et utiliser les « étiquettes » que l'on souhaite ;
- une fois qu'un jeu d'étiquettes est défini, même par quelqu'un d'autre, on peut l'étendre en y ajoutant ses propres créations.

Il ne faut en fait pas parler de langage XML au singulier, mais bien de langages au pluriel. En effet, XML désigne un certain type de fichier texte, respectant des règles d'écriture. À partir de ces règles, il est possible de définir de nouveaux langages, comme par exemple XHTML, SVG, Docbook, MathML...

À quoi ça ressemble ?

Exemple de fichier XML

Voici un petit exemple de fichier XML... Nous allons passer en revue ses différents constituants.

```
<?xml version="1.0" encoding="UTF-8"?>
<savants>
```

```
  <!-- Quelques physiciens importants -->
  <savant id="phys0">
    <identité prénom="Galileo" nom="Galilei"/>
    <dates naissance="1564-02-15" décès="1642-01-08"/>
    <contributions>Relativité du mouvement &amp;amp; système
héliocentrique</contributions>
  </savant>
  <savant id="phys1">
    <identité prénom="Isaac" nom="Newton"/>
    <dates naissance="1643-01-04" décès="1727-03-31"/>
    <contributions>Gravitation universelle, principe d'inertie,
décomposition de la lumière &amp;amp; calcul infinitésimal</contributions>
  </savant>
  <savant id="phys2">
    <identité nom="Einstein" prénom="Albert"/>
    <dates naissance="1879-03-14" décès="1955-04-18"/>
    <contributions><![CDATA[Relativités restreinte & générale, nature
corpusculaire de la lumière & effet photoélectrique]]></contributions>
  </savant>
</savants>
```

Dans le détail...

Bon nombre des notions que nous allons aborder seront familières aux habitués de HTML... Éléments, attributs, commentaires et sections CDATA sont couramment appelés des « *nœuds* ».

Le prologue

```
<?xml version="1.0" encoding="UTF-8"?>
```

Cette première ligne est le *prologue* du fichier. Il indique la version de la recommandation XML qui sert de base à l'écriture du fichier (il n'y a eu à cette date qu'une seule version, la version 1.0), ainsi que l'encodage de caractères utilisé. Cette ligne est facultative, mais cela ne coûte (presque) rien de l'ajouter... d'autant plus que cela permet de spécifier explicitement l'encodage. Par défaut, il s'agit de l'UTF-8.

Les commentaires

```
<!-- Quelques physiciens importants -->
```

Un commentaire, comme en HTML, commence par la chaîne de caractères `<!--` et se termine par `-->`. Tout ce qui est entre ces deux chaînes de caractères n'est pas une donnée du document, mais un commentaire laissé par le développeur et tout comme en HTML, la

visualisation du document XML dans un navigateur ne fait pas apparaître ce texte. Attention, il est interdit à l'intérieur d'un commentaire de placer la chaîne « `--` ».

Les éléments et attributs

Nous abordons là les étiquettes dont nous avons parlé en guise d'introduction. Ce sont les éléments et attributs qui permettent de qualifier les données, et de leur conférer un sens.

Les éléments

Il existe deux types d'éléments : les éléments vides... et ceux qui ne le sont pas.

- Un élément non vide commence par une balise ouvrante et se termine par une balise fermante. Dans le document précédent, des exemples de balises ouvrantes sont `<savants>` ou `<contributions>` ; les balises fermantes correspondantes sont respectivement `</savants>` et `</contributions>`.
- Un élément vide ne comporte qu'une seule balise, dite « auto-fermante ». C'est le cas de l'élément `identité` : il est caractérisé par la balise `<identité...` : vous noterez qu'elle se termine par `/>`.

Entre les balises ouvrante et fermante d'un élément non vide peuvent se trouver d'autres éléments. C'est le cas ici de la plupart des éléments, comme par exemple `savant` : l'indentation du code met en évidence la relation hiérarchique des éléments. Cette structure hiérarchique forme une arborescence.

Dans un document XML, un élément joue un rôle particulier : il s'agit de l'*élément racine*, qui doit être unique et contenir tous les autres éléments de l'arborescence. Ici, il s'agit de l'élément `savants` .

Les attributs

Quand on veut préciser ce que contient un élément, on peut placer un attribut sur la balise ouvrante. Par exemple, l'élément `date` comporte deux attributs, nommés `naissance` et `décès` . Un attribut possède un nom, ainsi qu'une valeur. Par exemple, l'attribut `prénom` du deuxième savant a pour valeur `Isaac` . Les attributs *ne sont jamais repris* dans la balise fermante : l'attribut `id` présent dans la balise ouvrante de l'élément `savant` est absent de la balise fermante `</savant>` .

Enfin, l'ordre dans lequel les attributs sont écrits dans une balise ouvrante n'a aucune importance : les attributs du dernier élément `savant` sont inversés par rapport aux deux premiers, mais les deux formes sont strictement équivalentes.

Les entités

Vous aurez noté que certains caractères ont une signification en XML : c'est le cas par exemple des caractères `<` ou `>`. Quand on doit les utiliser, il faut donc trouver une parade pour qu'ils ne soient pas interprétés. Pour cela, on utilise des *entités*, qui codent ces caractères par des chaînes de caractères les représentant ou des nombres. Une entité commence par le caractère `&` et se termine par un point-virgule `;`. Par exemple, le caractère `<` doit être écrit `<` ou `&#060;`. En particulier, le caractère `&` doit lui-même être codé comme une entité sous la forme `&`.

XML ne définit que cinq entités, au contraire de HTML où plusieurs dizaines existent. Les entités prédéfinies de XML sont :

- `<` pour <
- `>` pour >
- `&` pour &
- `&quote;` pour "
- `&aquote;` pour '

Les deux dernières entités peuvent par exemple être utilisées dans des valeurs d'attributs.

Les sections CDATA

Lorsqu'un contenu est susceptible de contenir plusieurs entités, il peut être fastidieux de les écrire. On utilise alors ce que l'on appelle une « *section CDATA* », qui est destinée à contenir des informations affichées telles quelles, sans traitement par l'outil de consultation (navigateur Internet ou tout autre type d'application chargée de traiter le document XML). C'est le cas dans notre exemple où l'on a écrit `<![CDATA[Relativités restreinte & générale, nature corpusculaire de la lumière & effet photoélectrique]]>` : il n'a pas été nécessaire de remplacer chaque occurrence de `&` par l'entité correspondante `&`.

Normes d'écriture

Un document XML doit répondre à un certain nombre de règles... sinon ce n'est pas un document XML. Ces règles sont les suivantes :

- Les majuscules et les minuscules sont différenciées et `toto` désigne un autre élément ou attribut que `Toto` ;
- Un nom d'élément ne peut pas commencer par un chiffre ;
- Si un nom d'élément n'est constitué que d'un seul caractère, alors ce caractère doit être une lettre ;
- Si le nom contient au moins deux caractères, le premier peut être un tiret « - » ou un tiret bas « _ ». Le nom peut ensuite être composé de lettres, chiffres, tiret « - » , tirets bas « _ » ou deux-points « : ».

- Tous les éléments doivent être fermés, et ce, dans l'ordre de leur ouverture : `<elt1><elt2>blabla</elt2></elt1>` est correct, mais `<elt1><elt2>blabla</elt1></elt2>` ne l'est pas.
- Les valeurs des attributs doivent être entre guillemets ;
- L'élément racine doit être unique.

Un document XML qui respecte ces quelques règles est dit « *bien formé* ».

Comment définir le contenu d'un document XML ?

Ce n'est pas parce qu'on sait comment écrire un document XML qu'on peut être sûr de savoir écrire un document similaire par la suite si le besoin s'en fait sentir. De plus, on peut souhaiter diffuser un type de document, ou bien être amené à travailler en parallèle sur un document donné. Pour ces raisons, il est nécessaire d'établir des règles communes qui définissent précisément les éléments et attributs autorisés, ainsi que leur hiérarchie -autrement dit, des règles de « grammaire ». Il existe trois grandes méthodes pour ce faire. Quand un document suit une telle grammaire et qu'il l'indique, il est dit « valide ».

Par une DTD

DTD signifie Document Type Definition, définition de type de document. Il s'agit du plus ancien langage qui permette de décrire un format XML (il est tiré de SGML, l'« ancêtre » de XML) et d'ailleurs c'est ainsi que les divers langages HTML ont été définis. Une DTD respecte une syntaxe particulière. Par exemple, pour définir un élément `élémentParent` qui contient un autre élément `élémentEnfant` au moins une fois et un attribut obligatoire `attr`, on écrit...

```
<!ELEMENT élémentParent (élémentEnfant)+>
<!ATTLIST élémentParent attr CDATA #REQUIRED>
<!ELEMENT élémentEnfant (#PCDATA)>
```

On peut déclarer que le document XML est conforme à cette DTD en insérant juste après son prologue une ligne qui spécifie la DTD et son type (publique, c'est-à-dire qu'elle est disponible pour tous sur Internet, système quand elle réside forcément sur la machine sur la laquelle le document XML est visualisé, ou interne quand elle est incorporée au document XML lui-même) :

- Pour une DTD système : `<!DOCTYPE élémentRacine SYSTEM "cheminVersLaDTD">`
- Pour une DTD publique : `<!DOCTYPE élémentRacine PUBLIC "identificationNormaliséeDeLaDTD" "URLDeLaDTD">`
- Pour une DTD interne : `<!DOCTYPE élémentRacine[Liste des déclarations...]>`

Voici par exemple la DTD de XHTML strict :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Une DTD permet aussi de définir de nouvelles entités réutilisables dans le document XML.

Par un XML Schema

Les DTD sont faciles à écrire et à comprendre, mais il ne s'agit pas d'un document XML. De plus, il y a quelques limitations comme le fait qu'il n'est pas possible de spécifier un type pour les données : il est par exemple impossible de forcer par une DTD un attribut à être un nombre entier strictement positif.

Le W3C a alors développé un langage XML qui permette de définir de nouveaux langages XML : le format XML Schema. Ce format permet de définir des types de données, et par rapport aux DTD offre de surcroît la possibilité de spécifier un nombre déterminé d'occurrences d'un élément donné, permet la mise au point de grammaires modulaires en facilitant l'importation d'un schéma dans un autre. Cerise sur le gâteau, il s'agit d'un langage XML : cela signifie qu'il est possible de le manipuler comme tout autre document XML. En contrepartie, le schéma XML équivalent à une DTD donnée est plus long. Dans le cas d'une grammaire particulièrement complexe, le schéma peut, si l'on n'y prend garde, devenir difficile à lire. L'équivalent de la DTD précédentes est ainsi...

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="élémentParent">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="élémentEnfant"
type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="attr" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

On déclare qu'un document XML est conforme à un schéma XML en utilisant un espace de nom XML (xmlns, XML NameSpace). Par exemple, pour indiquer qu'un document est en XHTML strict, on écrit

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Plus généralement, un espace de noms s'utilise ainsi :

```
<élémentRacine
xmlns:prefixe="http://www.mondomaine.org/schemas/monschema.xsd">
  <prefixe:élémentEnfant1...>
</élémentRacine>
```

Par un schéma Relax NG

La complexité relative du format XML Schema a amené le consortium Oasis à développer un format concurrent nommé Relax NG. Ce format reprend les mêmes types de données que le format XML Schema. Outre une syntaxe XML qui ressemble *grosso modo* à celle de XML Schema, Relax NG offre une syntaxe plus compacte se rapprochant de celle des DTD. Notre exemple récurrent donne dans cette syntaxe...

```
element élémentParent{element élémentEnfant {text}+, attribute attr
{text}}
```

En syntaxe XML, cela donne

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="élémentParent"
xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="élémentEnfant">
      <text/>
    </element>
  </oneOrMore>
  <attribute name="attr"/>
</element>
```

Transformer un document XML en... quelque chose d'autre avec XSL

Un document XML est un format de stockage de données. En tant que tel, il offre un grand intérêt s'il est possible de le manipuler afin de traiter ces données. Il est donc nécessaire de disposer d'un langage qui le permette.

Introduction

Le W3C a développé des technologies, le XSL pour eXtensible Stylesheet Language, qui est un format XML et est par conséquent manipulable et analysable de la même manière qu'un document XML « classique ». Ce langage permet la transformation d'un document XML en n'importe quel type de fichier texte -un autre document XML bien sûr, mais aussi de l'HTML qui ne soit pas du XHTML, du RTF, voire du PostScript ou tout autre format texte, voire PDF.

XSL consiste en trois grands modules : XPath permet de localiser des nœuds dans une arborescence XML, XSLT permet la transformation du document, et XSL-FO (pour Formatting Objects) en permet la mise en forme.

XPath

Il est nécessaire de disposer d'un format de description et de navigation au sein de l'arborescence du document XML source : il s'agit d'XPath. Ce langage est une sorte d'hybridation entre les instructions de navigation à taper en ligne de commande sous Windows ou Unix (comme `..` qui désigne le répertoire parent sous ces systèmes, et l'élément parent de l'élément en cours d'analyse sous XPath), et CSS (par exemple, pour cibler les éléments `elt` possédant un attribut `attr`, on écrit `elt[@attr]`). S'y ajoutent des instructions plus subtiles, ainsi que quelques fonctions mathématiques ou de manipulation des booléens. Le rôle de XPath est de pouvoir identifier un nœud quelconque, ou une collection de nœuds dans une arborescence XML, que ce soit dans l'absolu (c'est-à-dire en l'identifiant à partir de l'intégralité de l'arbre) ou en relatif (c'est-à-dire en partant d'un nœud déterminé au préalable).

XSLT

Principe

XSLT est un format XML défini à partir d'un schéma. Une « feuille de style » XSL, dont l'élément racine est `stylesheet` débute donc (après son prologue) par...

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
```

XSL définit un jeu d'instructions qui permettent le contrôle de la sortie.

Une fois que l'on dispose des moyens de pouvoir cibler des nœuds différents de l'arborescence, XSL nous offre la possibilité de la *transformer*. La transformation peut être simplissime et ne consister qu'en la recopie de la valeur du nœud dans le document de sortie, mais elle peut aussi être très complexe et mettre en œuvre des relations de dépendances entre différents nœuds. Pour l'anecdote, des mathématiciens ont démontré que comme le langage XSL permet de simuler une machine de Turing universelle, ce langage permet de mener à bien exactement les mêmes algorithmes que les langages de programmation tels que

Java, C# et autres C... évidemment, comme il n'est pas à l'origine conçu pour ce genre de tâche, cela risque d'être un peu compliqué ;-)

Par exemple, pour copier dans le document de sortie la valeur de l'élément `elt` fils de l'élément courant, on écrit `<xsl:value-of select="elt" />` . Cela est très simple ; mais XSL permet aussi de construire des tests `if... then...` ou bien des tests à valeurs multiples :

```
<xsl:choose>
  <xsl:when test="elt='bateau' or elt='canoë'">Ce moyen de transport
navigue.</xsl:when>
  <xsl:when test="elt='voiture' or elt='bicyclette'">Ce moyen de
transport roule.</xsl:when>
  <xsl:default>Ce moyen de transport sert à se déplacer dans l'air ou
dans l'espace.</xsl:default>
</xsl:choose>
```

XSL est un langage extrêmement riche, et il serait vain de prétendre en faire le tour en si peu de mots. Il suffit de retenir que ce langage permet de faire *ce que l'on veut* de tout document XML.

XSLT côté client

XSLT peut être interprété directement par le poste client. Je vais axer la suite de mon discours sur le cas particulier d'une transformation XML vers (X)HTML.

Si l'on utilise un navigateur Internet, on peut lui laisser la possibilité d'effectuer lui-même, dynamiquement, la transformation du document XML si on lui spécifie une feuille de style XSL... ou sa mise en forme si on lui fournir une feuille CSS. Pour cela, entre le prologue et l'élément racine, il suffit d'appeler la feuille de style avec ce que l'on appelle une « instruction de traitement » :

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="typeMIMEDeLaFeuilleDeStyle"
href="emplacementDeLaFeuilleDeStyle" ?>
<élémentRacine>
(...)
```

Il est en effet possible, si on fournit un feuille de style CSSen spécifiant le type `text/css` , d'appliquer cette feuille au document XML. Après tout, quand une feuille de style est appliquée à un document XHTML, elle est appliquée à un document XML particulier, et cela marche plutôt bien. Pour indiquer que les éléments `elt` doivent être écrits en rouge, il suffit d'écrire dans la feuille CSSla ligne `elt {color:red;}` .

Dans les deux cas, CSS ou XSL, la qualité de transformation ou de la mise en forme sont étroitement dépendants du navigateur utilisé. Vous savez déjà que le rendu CSSdiffère grandement d'un navigateur à un autre... eh bien, les transformations XSL sont encore plus variables. C'est la raison pour laquelle, à moins d'avoir un contrôle absolu sur les navigateurs utilisés sur vos postes client, il n'est pas recommandé de recourir à cette technique.

XSLT côté serveur

Une manière beaucoup plus fiable d'assurer un résultat final de transformation identique selon les navigateurs est de réaliser cette transformation côté serveur. Le navigateur (ou, plus généralement, l'application chargée du traitement) ne reçoit plus le document XML source, mais le résultat de sa transformation. Cela permet aussi de délivrer des résultats différents en fonction des requêtes réalisées depuis le poste client. Dans ce rôle, le document XML source se rapproche plus d'une base de données hiérarchique. Les langages de script incorporent soit nativement, soit par l'intermédiaire de bibliothèques des fonctions de manipulation des documents XML ainsi que les instructions XSL : c'est le cas de la bibliothèque libxslt pour PHP par exemple qui est implémentée par l'extension XSL.

XSL-FO

Ce langage permet de produire des documents au format PDF. XSL-FO donne un rendu très fin de la qualité visuelle du document produit au format PDF, au pixel près. Il permet en outre de pouvoir insérer des effets visuels impossibles à réaliser en (X)HTML/CSS, comme l'utilisation de polices de caractères particulières. Ces instructions ne sont pas supportées par les navigateurs Internet. On obtient une feuille de style XSL-FO en ajoutant à une feuille de style XSLT « normale » les instructions disponibles dans l'espace de nom -virtuel-http://www.w3.org/1999/XSL/Format. Les transformations XSL-FO sont la grande majorité du temps réalisées côté serveur, mais il est possible d'installer sur un poste client un utilitaire Java nommé FOP. Cet utilitaire est de surcroît capable de produire des documents de sorte aux différents formats texte, ainsi que PCL, AFP, PNG...

Conclusion

On le voit, dire que l'on fait du XML manque de précision : il s'agit plutôt d'**un ensemble de technologies et de langages qui permettent la définition, la manipulation, la transformation et le transfert de données entre applications**. Un cas particulier est la mise en œuvre dans le cadre d'un développement pour le Web, mais cela n'est que la partie émergée de l'iceberg. En fait, vos applications favorites manipulent déjà du XML sans que vous le soupçonniez, que ce soit la dernière version d'Office, Open Office, ou bien vos agrégateurs de contenus qui récupèrent des données aux formats RSS ou Atom, vos navigateurs Web qui traitent des documents XHTML...

XML permet de définir des langages plus ou moins complexes mais qui permettent d'établir des standards de formats de fichiers d'échanges entre applications. Peu importe le langage utilisé pour traiter des informations, s'il est capable de produire du code XML, il est capable de dialoguer avec tout autre programme écrit dans n'importe quel langage.
