Les formats de Fichiers

Table des matières

La sérialisation est un processus visant à coder une information sous la forme d'une suite d'informations plus petites. Cette suite pourra être utilisée pour la sauvegarde ou le transport sur le réseau. L'activité visant à décoder cette suite pour créer une copie conforme de l'information d'origine, s'appelle la désérialisation (ou unmarshalling).

Les termes marshalling et unmarshalling s'emploient le plus souvent dans le contexte d'échanges entre programmes informatiques, alors que les termes sérialisation et désérialisation sont plus généraux.

D'apparence simple, ces opérations posent en réalité un certain nombre de problèmes, comme la portabilité des encodages. Par ailleurs, les choix entre les diverses techniques de sérialisation ont une influence sur les critères de performances comme la taille des suites sérialisées ou la vitesse de leur traitement.

1. Format CSV

Les fichiers CSV (Comma Separated Values) sont des fichiers (texte) dans lesquels on écrit des données organisées par ligne. Ce type de fichier est très utilisé pour envoyer des données (par FTP, par email, etc.). Par exemple les résultats des tirages du loto (ordre des boules, gains, etc.) sont disponibles au téléchargement sous forme de fichier CSV sur le site de la Française des Jeux.

1.1. Le format

Le format des fichiers CSV est l'un des plus simples qu'on puisse imaginer. Les données sont organisées par ligne. Une ligne peut contenir plusieurs champs/colonnes séparés par des virgules. La première ligne donne (souvent) les titres des colonnes. Bien entendu, l'ordre des colonnes est le même dans toutes les lignes.

Par exemple :

Sexe,Prénom,Année de naissance
M,Alphonse,1932
F,Béatrice,1964
F,Charlotte,1988

représente les données suivantes :

Sexe	**Prénom**	**Année de naissance**
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988

Le fait que les fichiers CSV soient essentiellement utilisés autour de logiciels tableur, et que les séparateurs ne soient pas standardisés (virgules, points-virgules sous certaines localisations dont la française, etc.) rend ce format peu pratique pour une utilisation autre que des échanges de données ponctuels.

Les champs texte peuvent également être délimités par des guillemets. Lorsqu'un champ contient lui-même des guillemets, ils sont doublés afin de ne pas être considérés comme début ou fin du champ. Si un champ contient un signe utilisé comme séparateur (virgule, point-virgule, etc.), les guillemets sont obligatoires afin que ce signe ne soit pas confondu avec un séparateur.

1.2. La lecture

La lecture d'un fichier CSV est réalisée en trois étapes majeures :

1. la lecture bas niveau des données dans un fichier,
2. la prise en compte du format des données
3. l'utilisation (transformation) des données.

La première ligne des fichiers CSV est souvent une ligne de titre. Les éléments de cette ligne sont, grosso modo, les entêtes des colonnes. Il est donc intéressant pour notre lecteur CSV qu'il sache traiter ce type d'information. En outre, les fichiers CSV peuvent contenir des lignes vides et des commentaires dont il ne faudra donc pas tenir compte.

Il arrive très souvent qu'on ne connaisse pas à l'avance le bon séparateur à utiliser pour lire un fichier CSV. Les Anglais utilisent généralement la virgule, les Français préfèrent le point-virgule qui ne pose pas de problème avec le format des nombres, tandis que certains projets ont fait le choix du pipe ou de la tabulation, etc.

Il existe plusieurs manières de "deviner" le bon séparateur à utiliser. Dans tous les cas, il faut lire les lignes et tenter d'analyser les contenus. On peut ainsi parcourir toutes les lignes ou se contenter des N premières. Dans la mesure où un "vrai" fichier CSV peut être très volumineux, il est préfèrable de se concentrer sur les premières lignes seulement.

2. Format JSON

JSON (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Ces types de données sont suffisamment génériques et abstraits pour, d'une part, pouvoir être représentés dans n'importe quel langage de programmation, d'autre part, pouvoir représenter n'importe quelle donnée concrète.

Le principal avantage de JSON est qu'il est simple à mettre en œuvre par un développeur tout en étant complet.

Au rang des avantages, on peut également citer :

- peu verbeux, ce qui le rend lisible aussi bien par un humain que par une machine ;
- facile à apprendre, car sa syntaxe est réduite et non extensible (bien que ne souffrant que de peu de limitations) ;
- ces types de données sont connus et simples à décrire.

2.1. Le format

Un document JSON a pour fonction de représenter de l'information accompagnée d'étiquettes permettant d'en interpréter les divers éléments, sans aucune restriction sur le nombre de celles-ci.

Un document JSON ne comprend que deux types d'éléments structurels :

- des ensembles de paires nom / valeur ;
- des listes ordonnées de valeurs.

Ces mêmes éléments représentent trois types de données :

- des objets ;
- des tableaux ;
- des valeurs génériques de type tableau, objet, booléen, nombre, chaîne ou null.

Les chaînes de caractères sont entre " et les nombres sont non pas entre " et qui contiennent principalement des chiffres. Par exemple "123" est une chaîne de caractères et -.1e+3 est un nombre (c'est -100). Pour ce dernier, on a un peu exagéré en utilisant la notation *eX* qui correspond à 10 puissance X (comme sur les calculatrices).

Les listes de paires se composent de deux champs, le mot clé qui est un chaîne de caractères, suivi de : et d'une valeur qui peut être n'importe quoi. Les listes sont délimitées par des accolades. Par exemple, nous pourrions donner les propriétés d'un écran avec le *json* suivant :

```
{ "resolution_verticale": 1200, "resolution_horizontale": 900, "couleurs": true }
```

Json permet également de faire des tableaux :

```
[1200, 900, true]
```

On peut noter que c'est beaucoup plus compact, mais dans ce cas la position est importante, il faut savoir à quoi correspond le premier élément, le second... Un tableau est délimité par des crochets droits et les éléments sont séparés par des virgules. La beauté de *JSON* est que l'on peut faire des liste de tableaux, des tableaux de listes, des tableaux de tableaux, des listes de listes, de tableaux de listes de tableaux de listes...

Exemple :

```
{
    "nom" : "PlayList",

    "chansons" : [
        {
            "titre" : "Aces high",
            "auteur" : "Iron Maiden",
            "note" : 5
        },
        {
            "titre" : "On parole",
            "auteur" : "Sister Sin",
            "note" : 4
        }
    ]
}
```

JSON se base plus sur un modèle clé/valeur que sur un format de balisage. Cela permet d'éviter les balises de fermeture, la répétition et cela peut éventuellement représenter un gain de place sur de très gros fichiers (16 caractères économisés ici en comptant les espaces, soit 95% de la taille originale) par rapport au format XML.

2.2. La lecture

Les commentaires ne sont pas prévus par les spécifications JSON, cependant certains scripts d'analyse JSON prennent en compte les commentaires comme en JavaScript :

```
// Commentaires en fin de ligne

/*
commentaires en bloc
*/
```

Le fichier permet de charger de l'information stockée dans ce format à partir du serveur ou de transmettre au serveur de l'information dans un fichier de ce format, par exemple, le contenu d'un formulaire qui vient d'être rempli. Il y a donc trois aspects: le traitement par le navigateur, par le serveur, et la transmission des données entre les deux.

Les fichiers au format JSON s'utilisent dans différents langages de programmation, notamment PHP et Java grâce à des parseurs qui permettent d'accéder au contenu, et éventuellement de le convertir en classes et attributs, dans ce langage.

Le site json.org fournit un parseur en langage C et donne une liste de parseurs pour d'autres langages comme Python.

Exemple en langage Python :

```
import json
# En partant d'une donnée construite à partir de types de base
data = [
        {"nom": "PlayList",
         "chansons": [
                     {'titre': 'Aces high', 'auteur': 'Iron Maiden', 'note': 5},
                     {'titre': 'On parole', 'auteur': 'Sister Sin', 'note': 4}
                     ]
         }
        ]

# sauver ceci dans un fichier
with open("test.json","w") as json_output:
       json.dump(data, json_output)
# et relire le résultat
with open("test.json") as json_input:
       data2 = json.load(json_input)
```

3. Format XML

Le XML (Extensible Markup Language) est un langage informatique de balisage générique. Cette syntaxe est dite « extensible » car elle permet de définir différents espaces de noms, c'est-à-dire des langages avec chacun leur vocabulaire et leur grammaire.

Elle est reconnaissable par son usage des chevrons (< >) encadrant les balises. L'objectif initial est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes (interopérabilité).

XML s'est imposé comme format de référence pour l'échange de données, notamment de métadonnées. L'exemple d'un transfert d'informations entre base de données relationnelles permettra d'illustrer les avantages et limites de ce format pour cet usage.

L'exportation d'une table peut se faire en CSV. Mais ce format comporte vite des limites à grande échelle (Internet). Il n'est pas auto-documenté (encodage du texte, séparateurs, ordre et nom des colonnes ?). Il demande une documentation externe rarement automatisée entre les partenaires. Que faire lorsque les tables source et destination n'ont pas des structures identiques ? Pour cette raison, on peut préférer des échanges en SQL (à la fois langage de définition de données et langage de manipulation de données). Cependant, malgré de nombreux efforts de normalisation, SQL comporte beaucoup de risques d'incompatibilité entre les implémentations. XML est une solution plus robuste. On peut en constater l'efficacité sur Internet avec la syndication de contenu. Il n'y a pas d'exemple connu d'échange de métadonnées réparties sur autant de « clients » et de « serveurs ».

3.1. Le format

Un document XML doit comporter un ou plusieurs éléments. Il y a exactement un élément appelé élément racine ou élément document, dont aucune partie n'apparaît dans le contenu d'un autre élément. Le nom de la balise de fin d'un élément doit correspondre à celui de la balise de début. Les noms tiennent compte des majuscules et des minuscules.

Si la balise de début figure dans le contenu d'un autre élément, la balise de fin doit également figurer dans le contenu du même élément. Plus simplement, les éléments délimités par les balises de début et de fin doivent s'imbriquer correctement les uns dans les autres.

La fin de chaque élément commençant par une balise de début doit être indiquée par une balise de fin comportant le même nom que celui utilisé dans la balise de début. Le texte figurant entre la balise de début et la balise de fin est appelé le contenu de l'élément. Un élément sans contenu prend la forme spéciale suivante : <nom/> . La barre oblique devant le caractère > remplace la balise de fin.

Les noms d'éléments peuvent comporter des lettres, des chiffres, des tirets, des traits de soulignement, des deux-points ou des points. Le caractère deux-points (:) ne peut être utilisé que dans le cas particulier où il sert à séparer des espaces de noms. Les noms d'éléments commençant par xml, XML ou une autre combinaison de la casse de ces lettres sont réservés à la norme XML.

Les caractères < et & ne peuvent pas être utilisés dans le texte, car ils sont utilisés dans le balisage. Si vous devez employer ces caractères, utilisez < à la place de < et < à la place de &. Les caractères >, " , et ' peuvent également être remplacés par > , " et ' respectivement.

Des commentaires peuvent figurer n'importe où dans un document en dehors des autres balises. Un processeur XML peut permettre à une application, sans que cela soit une obligation, d'extraire le texte des commentaires. La chaîne de caractères "--" (deux tirets) ne doit pas figurer à l'intérieur des commentaires.

Les documents XML doivent commencer par une déclaration XML qui précise la version de la norme XML utilisée.

Exemple 1 : fichier users.xml des comptes de connexion des internautes

```
<?xml version="1.0" encoding="UTF-8" ?>
<users>
    <user>
        <id>1</id>
        <login>Gil</login>
        <password>1234</password>
        <website>projet.eu.org</website>
        <created_at>2014-01-16 23:04:46</created_at>
    </user>
    <user>
        <id>2</id>
        <login>Dom</login>
        <password>4321</password>
        <website>promethee.eu.org</website>
        <created_at>2014-01-16 23:04:47</created_at>
    </user>
</users>
```

Exemple 2 : fichier messages.xml des commentaires laissés par les internautes

```
<?xml version="1.0" encoding="UTF-8" ?>
<messages>
    <message>
        <id>1</id>
        <user_id>1</user_id>
        <title>Bonjour</title>
        <body>Un bonjour de Paris ;)</body>
        <created_at>2014-01-16 23:04:48</created_at>
    </message>
    <message>
        <id>2</id>
        <user_id>2</user_id>
```

```
        <title>Super site</title>
        <body>J'apprécie ton site, continue ainsi !</body>
        <created_at>2014-01-16 23:04:49</created_at>
    </message>
    <message>
        <id>3</id>
        <user_id>2</user_id>
        <title>Merci pour tout</title>
        <body>J'oubliais de dire que ton site m'a beaucoup servi !</body>
        <created_at>2014-01-16 23:04:50</created_at>
    </message>
</messages>
```

3.2. La lecture

Avant d'entrer dans un processus XML, un contenu doit être « xmlisé ». Cette opération est effectuée par un processeur XML. Les parseurs les plus répandus sont :

- libxml2 - Le processeur XML libre du système d'exploitation GNU/linux, accessible en C, Python10, PHP11, et en Ruby12
- Xerces - XML Java Parser, le parseur XML par défaut d'une machine virtuelle Java, accessible en Java
- Expat - Le parseur XML de James Clark, notamment embarqué par les navigateurs mozilla (firefox).
- …

Il en existe beaucoup d'autres, en particulier en Java, adaptés à différents cas particuliers : ouvrir une API plus simple, accepter des documents mal formés comme HTML, traitements plus simples (notamment pour les documents longs).

Une fois « xmlisé », un document est accessible à différents langages, selon des interfaces de programmation standardisées.

Exemple : application PHP permettant de retrouver les messages de Dom

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US" lang="en-US">
<head>
    <title>Messages de Dom (source : XML)</title>
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<?php

function display($string)
{
    echo htmlentities(utf8_decode($string), ENT_QUOTES, 'ISO-8859-1');
}

$xml_users    = file_get_contents("users.xml");
$xml_messages = file_get_contents("messages.xml");

$src_users    = simplexml_load_string($xml_users);
```

```
$src_messages = simplexml_load_string($xml_messages);

$users = $src_users->xpath("/users/user[login = 'Dom']");
$dom   = $users[0];

if ( $dom )
{
    ?>
    <table border="1">
        <tr>
            <th>login</th>
            <th>title</th>
            <th>text</th>
        </tr>
        <?php
        $messages = $src_messages->xpath("/messages/message[user_id = '".$dom->id."']");
        foreach($messages as $message)
        {
            ?>
            <tr>
                <td><?php display($dom->login); ?></td>
                <td><?php display($message->title); ?></td>
                <td><?php display($message->text); ?></td>
            </tr>
            <?php
        }
        ?>
    </table>
    <?php
}

?>
</body>
</html>
```