Cette note de synthèse a pour objet de décrire le fonctionnement et l'intérêt des différents parseurs XML. Nous allons développer cette synthèse en 4 grandes parties. Dont voici la première.

Quel est l'intérêt des fichiers XML ?

Il y a de nombreux intérêts à utiliser les fichiers XML et le langage XML en général.

- XML est une norme qui est maintenue par le W3C, une organisation qui est responsable des normes sur toutes les applications et les sites internet. Les documents XML sont neutres, ils ne sont pas liés à une application ou à une entreprise. Il existe de nombreux formats de documents disponibles pour les programmeurs. Ces types de formats sont souvent propriétaire et ne peuvent être créées et lus uniquement grâce à un logiciel qui est compatible avec ce format. A l'inverse des documents XML peuvent être créés dans un des nombreux éditeurs XML. il est possible de créer un document XML dans un éditeur puis de le modifier dans un autre sans problèmes de compatibilités. Les documents XML peuvent également être créés dans un éditeur de texte tel que le Bloc-notes (Bien que ça ne soit pas recommandé).

- XML est de par son nom extensible (Extensible Markup Language), il utilise des éléments ou des balises pour définir la structure du document. En définissant sa structure, il est possible d'utiliser des outils extérieurs tels que les feuilles de style pour manipuler et réutiliser ce contenu. En séparant le contenu de l'affichage, il est possible d'utiliser une seule source de contenu et l'intégrer dans de nombreux contextes différents. XML n'a pas un nombre fixe de balises ou d'éléments, comme en HTML, mais il est extensible, il permet donc au développeur de définir les balises significatives. En utilisant XML, les développeurs peuvent créer un langage de balisage qui est adapté à leur but. Les développeurs peuvent définir de nouveaux éléments dont ils ont besoin pour leurs besoins spécifiques. Cette capacité à définir des éléments personnalisés rend XML extrêmement polyvalent.

- Il est également possible de réutiliser le contenu des documents XML ce qui permet de rendre les développeurs plus efficaces. XML soutient et encourage ce genre de processus en étant flexible et modulableg. Il est possible de créer du contenu, puis ré-utiliser ce contenu dans différents documents. Les documents XML peuvent être manipulés en fonction des besoins des différents utilisateurs. Il est assez facile d'appliquer des feuilles de style pour un document XML afin de manipuler le contenu.

- XML permet de séparer le contenu du format. La mise en forme du document XML est à l'intérieur d'une feuille de style séparée. Cette séparation permet de maintenir facilement et de mettre à jour la mise en forme au fur et à mesure que les besoins changent.

- XML est un des éléments clés du langage Web, en effet il est très proche de par sa syntaxe au HTML et donc s'intègre très facilement dans les documents HTML et par conséquent dans les sites web.

Les différences entre les deux familles de processeurs SAX/DOM ?

Les deux processeurs DOM et SAX sont largement utilisés pour lire et analyser les fichiers XML dans les applications, les deux ont leurs propres avantages et d'inconvénients.

1. PROCESSEUR DOM

Le processeur DOM est une API (Application Programming Interface) arborescente. Une API arborescente est centrée autour d'une structure et fournit des informations sur les composants d'un arbre.

Un processeur DOM crée une structure arborescente en mémoire à partir du document d'entrée, puis attend les demandes du client. Un processeur DOM fourni toujours à l'application cliente l'ensemble du document, peu importe la quantité d'information réellement nécessaire par le client. Avec les processeurs DOM, les appels de méthode doivent être explicites pour ne pas envoyer sans cesse le document à l'application cliente.

2. PROCESSEUR SAX

SAX est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML. Une application utilisant SAX implémente généralement des gestionnaires d'événements, lui permettant d'effectuer des opérations selon le type d'élément rencontré.

Prenons pour exemple ce document XML :

```
<Personne>
<Nom>Ducoulombier</nom>
<prenom>Simon</prenom>
</Personne>
```

Une interface événementielle telle que l'API SAX permet de créer des événements à partir de la lecture du document ci-dessus. Les événements générés seront :

```
start document
start element: personne
start element: nom
characters: Ducoulombier
End element: nom
start element: prenom
characters: Simon
End element: prenom
End element: personne
End document
```

Ainsi, une application basée sur SAX peut gérer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document.

3. DIFFERENCE ENTRE DOM ET SAX

DOM (Document Object Model)

- Fourni le document entier à chaque appel de fonction
- Représente le résultat comme un arbre

- Permet de rechercher dans l'arbre
- Permet de modifier l'arbre
- Bon pour la lecture de fichiers XML

SAX

- Analyse jusqu'à ce que vous lui dise d'arrêter
- Ne renvoie pas tout le document XML à chaque appel
- API de bas niveau
- Bon pour les très grands documents, surtout il faut accéder à de très petites portions du document.

4. COMMENT CHOISIR ENTRE DOM ET SAX PARSERS?

Idéalement, un bon analyseur doit être rapide, riche en fonctionnalités et facile à utiliser. Mais en réalité, aucun des principaux processeurs ne possèdent tous ces avantages. Par exemple, un processeur DOM est riche en fonctionnalités (car il crée un arbre DOM en mémoire et permet d'accéder à une partie du document à plusieurs reprises, et permet de modifier l'arborescence DOM), mais il prend beaucoup de place en mémoire lorsque le document est énorme, et il faut un peu de temps pour apprendre à travailler avec lui.

Un processeur SAX, cependant, est beaucoup plus efficace pour l'espace mémoire dans le cas de gros document d'entrée (car elle ne crée pas de structure interne). De plus, il fonctionne plus vite et est plus facile à apprendre que le DOM parce que son API est plus simple. Mais du point de vue de la fonctionnalité, il offre moins de fonctions ce qui signifie que les utilisateurs eux-mêmes doivent s'occuper de plus de chose, comme la création de leurs propres structures de données.

La réponse dépend vraiment des caractéristiques de l'application et des exigences actuelles.

5. PEUT-ON UTILISER SAX ET DOM EN MEME TEMPS ?

Oui, bien sûr, parce que l'utilisation d'un processeur DOM et d'un processeur SAX est indépendant. Par exemple, si l'application a besoin de travailler sur deux documents XML, et fait des choses différentes sur chaque document, il est possible d'utiliser un processeur DOM sur un document et un processeur SAX sur un autre, puis combiner les résultats.

3. DOM

1. GENERALITES

Les processeurs DOM se basent sur un mécanisme standard du W3C.
Le format DOM représente un document XML dans son intégralité pour pouvoir l'exploiter entièrement en une seule fois.
Par le biais de l'API DOM, qui peut être implémentée différemment selon le langage utilisé, l'utilisateur peut rechercher le document, le modifier, ajouter des éléments ou en supprimer.

Un document XML est construit de cette façon par un processeur DOM orienté objet :

- La racine du document est la racine de l'arbre des éléments
- Tous les éléments qui le composent sont des objets

- Chacun de ces objets hérite du type Node

Les propriétés que l'on peut retrouver pour construire le type Node sont les suivantes :

- Nom
- Valeur
- Noeud parent
- Noeuds fils
- Attributs

2. Aspect orienté objet

Pour comprendre les conséquences de cette architecture, il faut s'intéresser au principe de l'héritage.
On doit pouvoir construire un arbre avec tous les éléments de notre document. Il faut donc que tous les attributs de chaque sous-objet de Node soient communs afin de ne pas devoir tester le type de chaque élément. En définitive, il faut intégrer tous les attributs nécessaires à chaque sous-objet dans l'objet Node.
L'inconvénient de cette structure est que les propriétés de certains sous-objets de Node ne sont pas utiles à tous les autres.

```
<boisson>
        <type>bière</type>
        <marque>Duvel</marque>
        <contenanceL>0.33</contenanceL>
</boisson>
```

Portion d'un document XML

Ici, "<marque>" est représenté par une instance d'un sous-objet de Node. Sa propriété "text" a pour valeur "Duvel".
"<boisson>" est aussi représenté par une instance d'un sous-objet de Node. Sa propriété "text" n'a pas d'intérêt.

```
public interface INode {
        public String getName();
        public String getValue();
        public Map<String,String> getAttributes();
        public Node getParent();
        public List<Node> getChildren();

        public void addChild(Node n);
        public void removeChild(Node n);
        public void hasChild(Node n);
        public int getChildrenLength();
}
```

Représentation partielle en Java de l'interface INode que la classe Node pourra implémenter

3. Organisation des specifications

Jusque récemment, les spécifications du DOM étaient éditées ponctuellement. Les versions qui ont été créées étaient appelées des levels.
Trois levels ont été établis avant que le working group chargé du DOM ne décide d'en faire un "living standard", c'est-à-dire un unique document qui évolue pour intégrer de nouvelles spécifications. Ce document est hébergé à l'adresse https://dom.spec.whatwg.org/.

- Level 1

Cette spécification documente les bases du DOM, c'est-à-dire les interfaces nécessaires pour représenter et manipuler un document structuré. Il inclut également des spécifications avancées et extensibles pour les documents XML. On y trouve par exemple les interfaces Document et Node.

- Level 2

Elle offre entre autre de nouvelles interfaces pour le XML, permet d'accéder et mettre à jour dynamiquement la représentation d'un document, ajoute la gestion d'événements et la manipulation de feuilles de style.

- Level 3

Elle permet de charger un arbre DOM depuis un document XML et inversement. Elle offre aussi la possibilité de modifier des documents en respectant un format valide, elle ajoute la gestion d'événements clavier, et fournit des méthodes de parcours d'un arbre DOM grâce à XPath.

Il y a donc encore eu des modifications depuis la rédaction du level 3. À ce jour, la dernière édition date du 10 Juin, preuve que le Document Object Model est toujours actif et suivi.

4. Lien avec les cours de structure de données

Le modèle DOM permet une représentation sous forme arborescente d'un document XML.
Comme nous l'avons vu en cours, les arbres sont un format de représentation des données particulièrement utilisé en informatique. D'une part pour leur simplicité de représentation, mais aussi pour l'efficacité des algorithmes que l'on peut appliquer.
Les arbres binaires de recherche, par exemple, offrent un excellent moyen de stocker et manipuler des données, avec une complexité de l'ordre de log(n).

Le fait de pouvoir représenter un document XML comme un arbre DOM offre les avantages des structures arborescentes. Toutefois, pour des fichiers de taille conséquente, il faut prendre en compte l'impact que de tels traitements ont sur la mémoire de la machine. Car même si la puissance de calcul est largement optimisée pour manipuler un fichier de ce type, on peut être limité par la mémoire disponible.
On ne peut donc pas négliger l'intérêt des processeurs SAX, qui sont beaucoup moins gourmands en ressources malgré leurs fonctions limitées.

Sources :

https://developer.mozilla.org/fr/docs/DOM_Levels
https://en.wikipedia.org/wiki/Document_Object_Model
https://en.wikipedia.org/wiki/XML
https://www.w3.org/TR/dom/
http://www.w3schools.com/xml/