I. Modalité d'évaluation et programme

EVALUATION

- 40% compte rendu de projet
- 60% DST type DST de RT0704

PLAN

- Moyen de représentation des données : XML
  - Un format
  - Des outils standardisés
- Gestion et automatisation des projets : Maven ou Gradle
- Un langage : Jakarta Java (JEE)
  - Framework : Spring Boot
- Retour sur les WEB services : après FLASK, la norme SOAP

II. XML, Extended Markup Language

Un standard du W3C, organisme de normalisation du web. W3C à pour but de normaliser la norme XML pour garantir la communication et l'échange des données lors de son utilisation.
XML est indépendant des langages, des plateformes et des logiciels.
Il se définit entre un format et un langage.
XML permet de :

- Formater des données
- Echanger des données
- Une fois les données extraites il n'y a pas à parser les données.

XML est à privilégier à JSON quand on souhaite plus de contrôle sur les données. Cependant, XML est beaucoup plus verbeux que JSON, il a donc des plus gros fichiers.

1. Structuration des données

Structurer les données est nécessaire. La structuration des données est dépendante de l'utilisation. Il n'existe pas de solution unique. Le but est de limiter le surtraitement des données. Il faut donc éviter les splits, les recherches, dans des champs qui pourraient être séparées dès le départ.
La structure permet de faire le contrôle de connaissance. Sa structure est donc extrêmement stricte.
Il est possible de structurer les types : *exemple :* la date de naissance doit être de tel type.
La première opération doit donc être le contrôle de la cohérence pour être sûr de récupérer les bonnes données.
La deuxième opération doit être l'automatisation des traitements. Grâce au contrôle de l'étape précédente il est plus facile d'automatiser les traitements.
Enfin, XML permet de normaliser les échanges.

XML a une représentation balisée. HTML est un dialecte, une extension, d'XML. XML est extensible.
Le but d'XML est d'organiser les données, ce n'est pas un format de visualisation ou de programmation.

La première ligne d'un document XML permet :

- De donner l'encodage
- De savoir si le document vient seul ou pas
- De donner la version

Il est possible de rajouter des paramètres dans les balises.

Attention, entre deux balises les espaces sont interprétés comme faisant partie de la chaîne de caractère.
Il n'y a pas de règles prédéfinies sur ce que l'on décide de mettre entre des balises ou en paramètre de celles-ci.

Nœud XML : une tabulation.

2. Contrôle de structure

1. Vérification **syntaxique** :
    a. Tous les attributs doivent être entre guillemets car rappel :
        i. Un simple guillemet n'est pas interprété
        ii. Un double guillemet est interprété (*exemple :* remplace le contenu par une valeur)
    b. Toutes les balises ouvrantes doivent être fermées
    c. Toutes les balises doivent être écrites en minuscules
    d. Les balises uniques doivent avoir la forme suivante : `<balise />`

2. Vérification **applicative** : permet d'éviter les erreurs des traitements et la transmission des données.

Le document doit être :
1- Bien formé (Cf. vérification syntaxique)
2- Valide : structure explicitement définie
    a. DTD : format de contrôle de la structure des documents XML mais qui n'est pas en XML. Il est déprécié car remplacé par « Schéma ». Il définit :
        i. Les balises
        ii. Le type de données :
            1. `#PCDATA` : données à analyser
            2. `#CDDATA` : données à ne pas analyser
            Exemple : 17x + 3 < 4x + 4 ← il ne faut pas qu'il analyse le « < » car il pensera que c'est une balise.
        iii. Les éléments de contrôle :
            1. `?0` ou `1`
            2. `+` : au moins un élément dedans.
            3. `*` : nombre indéterminé, 0 compris.
        iv. Définition de l'obligation de présence, ou non, d'un attribut :
            1. `#IMPLIED` : optionnel
            2. `#REQUIRED` : obligatoire

        La limite des DTD est qu'il n'y a pas de contrôle de contenu. C'est un format qui est limité à des architectures simples.

    b. Schéma : un schéma est un document XML qui décrit un document XML.
        Déclaration du schéma :
            On définit le namespace : « `xmlns:xs` ». Pour rappel, ue namespace sert à donner un environnement et à dire à quoi on se réfère.
            `<xs:schema xmlns :xs='http://www.w3.org/2001/XMLSchema'>` → inique le namespace et averti que tout ce qui suit est un schéma qui respecte la norme.

        Elément du schéma :

Le schéma permet de définir les éléments, les attributs, les valeurs par défaut, …, qui apparaissent dans un document. Les attributs ont donc un type et des contraintes. Le but est de laisser le moins d'interprétation possible, d'éviter les ambigüités.

Exemple : `<xs :element name='le_nom' type='le_type' fixed= 'val_def' />` ← le dernier champ est un exemple de contrainte, il permet de fixer une valeur de contrôle.

Les éléments complexes, `complexType` : balise qui contient des autres balises et/ou d'autres attributs, des séquences.

Il est possible de donner des zones de restrictions, d'enlever les espaces inutiles (*exemple :* `<xs:whitespace value= 'collapse'>`).

Intégration au document :

Le définir dans la balise racine en le référençant avec la balise « `xsi:schemaLocation` ».

La validation se fait par :

- Un éditeur : `xmlspy.eclipse`.
- Un navigateur
- Une ligne de commande : `xmllint : xmllint --noout --schema video.xsd ./video.xml`
  Cette option permet de ne pas réimprimer le fichier.

20/03/2023

3. Accès aux éléments : XPATH

(Il existe également XQuery, mais intéressant quand on a des très gros volumes de données. Il a été créé pour éviter de créer des bases de données relationnelles. Il y a moins d'intérêt aujourd'hui avec l'apparition de MongoDB.)

XPATH est un langage de requêtes simples, normalisé par le W3C. Il est possible d'utiliser XPATH dans n'importe quel langage.
Il fonctionne par l'exploitation de « chemin » pour naviguer dans les documents. Une expression peut désigner un ensemble d'informations.

a. Terminologie

Nœud : tout élément que l'on trouve dans un document XML (balises et contenus).
Racine : première balise, ce qui est au niveau 0. Il n'y a qu'une seule balise au niveau 0.
Elément : valeur associée à un nœud.
Attributs

Les relations entre les nœuds dépendent de l'architecture et de l'environnement. Les différentes relations sont :

- **Parent** : balise du niveau directement au-dessus en termes de niveau. Un niveau.
  Exemple :
  ```
  <carnet>
        <personne>
  ```
  Le parent de « personne » est « carnet ».
- **Children :** balise du niveau directement au-dessous en termes de niveau.
- **Siblings** : balises qui sont au même niveau.

- **Ancestros** : du nœud jusqu'à la racine.
- **Descendants** : du nœud jusqu'aux « feuilles ».

b. Format XPATH

Les expressions peuvent être rentrées en relatif ou en absolue.
Le résultat peut être :

- Un ensemble de nœuds : sous forme d'une liste sans itérateurs.
- Une chaîne de caractère
- Un nombre
- Un booléen

Notations :

- `//` : sélection depuis le nœud courant de tous les nœuds qui respectent l'expression
- `@` : attribut
- `[]` : accès à une valeur.

Fonctions :

- `count()` : comptage du nombre d'éléments
- `name()` : nom du nœud
- `text()` : contenu d'un PCDATA
- `last()` : sélection du dernier élément
- `position()` : position du nœud courant dans la sélection
- `normalize-space()` : suppression des espaces de début et de fin de chaîne.
- `string(num)` : transforme un nombre en chaîne
- `string-lenght(string)` : longueur d'une chaîne de caractères
- `contains(string, string)` : test d'inclusion
- `substring-before/after(string, string)` : test la sous chaîne
- `upper/lower-case(string)` : modification de la casse

Fonctions pour se déplacer :

- `descendant` : exclu le nœud en cours
- `descendant-or-self` : noeud du sous arbre ou le nœud courant.
- `parent`
- `ancestor`
- `ancestor-or-self`
- `following-sibling`
- `preceding-sibling`

Exemples :
`/video/film/titre` → affiche une liste de pointeurs
`/video/film/titre/text()` → affiche la liste des titres
`/video/film[2]/titre/text()` → affiche le titre du 2ième film. Attention ne commence pas à 0.
`/video/film[@annee=1992]/titre` → affiche une liste de pointeurs vers des nœuds titres datant de 1992.
`count(/video/film)` → compte le nombre de films
`/video/film[2]/preceding-sibling ::film/titre/text()` → affiche le titre du film precedent le film.

Exercice :

1. Noeud du réalisateur de Pale Rider
   `/video/film[normalize-space(titre)='Pale Rider']/réalisateur/`
2. Liste de tous les acteurs présents dans le fichier
   `//acteur/text()` → mais affiche les acteurs plusieurs fois si un acteur est dans plusieurs films.
3. Nombre d'acteurs présents dans le fichier
   `count(//acteurs)`

4. Acteur d'index 2 du film suivant celui d'index 2
   `video/film[2]/following-sibling::film/acteurs/acteur[2]/text()`
5. Nombre de films
   `count(/video/film)`

c. XPATH et Java

Chaque langage a une API qui permet de gérer du XML.

Validation

- Pendant le parsing : validation automatique, de la syntaxe, au moment de l'exploitation du document.
- Validation manuelle : en amont contrôle des données.

La classe nœud permet de récupérer un objet nœud. Il existe 12 types/héritages définis de cette classe.

Création du document

On souhaite un objet XML en sortie. Différentes solutions :

1- `DocumentBuilderFactory` ← permet de fabriquer un objet
`factory.setNamespaceAware(true)` ← Le factory doit pouvoir gérer les namespaces.
…
```
Document doc = builder.parse(xmlFile) ;
```

2- `parse( File f)`
`parse(InputStral is)`

3- Parser une chaîne de caractères :
```
String xmlString = « <nn> … </nn> » ;
Document doc = builder.parse( new InputSource( new
StringReader(xmlString)));
```
L'objet `doc` est un document XML.

Faire une « requête » avec XPATH en Java :

Exemple :

```
expression = « /video/film/titre/text() »
expr = xpath.compile(expression) ;
result=expr.evaluate(doc, XPathConstants.NODESET);
nodes = (NodeList) result ;
```

Le texte est un nœud. En effet, la valeur est un paramètre de la classe nœud.

4. Utiliser XML

a. Parser

Parser : processus de structuration des données. Il assure le lien entre le fichier et le programme. Il est associé à une méthode de traitement (lecture et écriture).
Les fichiers XML sont souvent créés automatiquement, ils ne sont pas créés à la main.
Pour créer ces fichiers il ne faut pas créer des chaînes de caractères pour créer un fichier final. Il faut utiliser des outils XML pour créer ces fichiers.

Pour parser il existe deux méthodes principales :

1- DOM (Document Object Model) : cette méthode modélise le fichier XML comme un arbre. C'est la méthode le plus ancien. Il permet de tout faire :
   a. Ajouter des éléments (un sous arbre)

   Le problème est qu'il charge l'intégralité du document en mémoire.
2- SAX (Simple Api for XML) : cette méthode modélise le fichier XML comme une suite d'événements. Un « œil » lit le document et lève des éléments (balise…) au fur et à mesure.

Association des événements à des `callback`. Aucune prise sur la lecture. Cette méthode s'approche de la programmation événementielle.
Son inconvénient est qu'il n'est pas possible de rajouter des données. Elle est seulement faite pour parcourir les documents XML.
SAX s'articule autours de différentes classes. Il implémente un handler par défaut, `contenthandler`, qui gère :

a. La gestion des débuts et fin de document
   `startDocument()`
   `endDocument()`
b. La notification de début d'élément
   `startElement(String namespaceURI, String localName, String qName, Attribute atts)`
c. La notification de fin d'élément

Un caractère est une donnée particulière.

3- (StAX Streaming Api for XML : le document est perçu comme un flux, un buffer que l'on charge à la demande. La différence avec SAX est qu'on peut revenir en arrière, avancer. Il permet également d'écrire / d'insérer dans le buffer. La problématique est la gestion du buffer.)

API :

- **JAXP** (API d'Oracle) : contient XPATH, XSLT, DOM, SAX et StAX.
- **Xerces :** contient XPATH, XSLT, DOM, SAX et StAX.
- **Sablotron** : XPATH, XSLT, DOM
- **Jdom** : API simplifiée DOM. API indépendante pour le moment.

SAX (Simple API for XML)

SAX fait une lecture séquentielle. SAX permet d'avoir accès au contenu du document mais ne donne pas accès au document. On ne peut pas ajouter / modifier / supprimer des balises avec SAX.
SAX est utilisé avec DOM pour construire une architecture en mémoire. Les choses importantes sont les différents caractères, pas ce qu'il y a entre les balises. Quand il n'y a aucune référence à un schéma tout ce qu'il lit est **de la chaîne de caractères.** SAX est simple à mettre en place mais devient plus compliqué quand on a de nombreuses balises différentes.

Le document XML est une série d'événements. Il faut donc regarder si une balise nous intéresse ou pas. Cela est géré avec un gestionnaire d'exception qui est mis en place par le langage. Les exceptions peuvent être adaptés avec un mécanisme d'héritage. Le principe est donc qu'il faut créer une chaîne de caractères et choisir à quel moment on commence à regarder et quand on arrête.

Le `ContentHandler` donne la liste des événements :

- `startDocument()`
- `endDocument()`

Il faut hériter `ContentHandler` et surchargé les deux fonctions précédentes, avec `@Override`, pour dire ce que l'on souhaite faire au début et à la fin. Il y a le même principe avec les événements :

- `startElement(…)`
- `endElement(…)`

Les paramètres de ces fonctions sont rajoutés directement (principe des événements).

Cf. slide 13 pour un exemple de fonction qui permet de récupérer une chaîne de caractère qui nous intéresse.

`characters(…)` → permet de récupérer les caractères.
`StringBuffer(…)` → chaîne de caractères modifiable.

Cf. slide 14-15 pour un exemple lié au traitement associé aux balises ouvrantes.

Il est capable de récupérer la valeur d'un attribut en entier puisque les valeurs sont entre guillemets. Il récupère toujours une chaîne de caractère donc, si c'est un entier, on utilise `Interger.parseInt(…)`.

On définit une classe d'analyse.

DOM

Le document XML est perçu sous la forme d'un arbre.
La gestion des axes est le fait de sa déplacer dans l'arbre.
On a accès au contenu du document mais également à la structure. On peut ajouter / modifier / supprimer des balises avec DOM.
DOM est une API minimale mais on utilise généralement 20% de l'intégralité de cette API.
Il n'y a pas d'implémentation imposée.
DOM a 3 niveaux :

- Niveau 1 : HTML… au niveau du noyau
- Niveau 2 : CSS, namespace…
- Niveau 3 : les schémas, les entrées et les sorties (écrire et lire sur le disque → je vais chercher la chaîne de caractère et il se débrouille pour trouver l'objet XML).

Il y a différents éléments qui sont nécessaires à la gestion des documents :

- NodeList
- Node
- CDATASection : éléments non interprétés
- …

Ce sont toutes les classes que l'on exploite dans la gestion. Dans ses classes on trouve un ensemble d'éléments, *cf.* la documentation Java.

Les nœuds sont des objets XML présents dans le document. En XML **on ne se contente pas des balises et de leurs contenus**. Le nœud est l'élément de base que l'on manipule. A l'usage on doit savoir ce que ce nœud est (caractère mort, attributs, …). Pour cela il existe 12 types d'objets *cf. slide 25* pour voir la liste.
On peut donc agir sur les nœuds :

- Le créer
- Le modifier
- Le supprimer
- L'extraire

L'élément est une structure balisée. L'élément est vu dans son ensemble. Un élément n'a pas de valeur associée. Il faut rentrer dans le Nœud Contenu pour récupérer la valeur associée.
L'idée est d'éviter de mélanger le texte et les balises.
Cf. slide 14 il aurait plutôt fallut faire :

```
<normal> Voici mon </normal>  <bold>livre</bold> <normal> ! </normal>
```

Cela multiplie le nombre de balise mais l'utilisation sera plus simple.

Conclusion on manipule ici des nœuds. La classe `NodeList` est une collection ordonnée de nœuds. Les listes ne sont pas itérées ce qui permet d'implémenter ce principe dans n'importe quel langage.

Etapes du DOM :

1- Définition et création du parser *Cf. exemple slide 29.*

2- Parcours d'un document *Cf. exemple slide 30.*

La racine est un élément, mais avec la logique objet il récupère un Nœud. **On manipule seulement des nœuds.**
`xxx.item(i)` → renvoie une référence sur un Node donc on caste avec `(Element)` pour ne travailler que sur des éléments par la suite : `(Element) films.item(i)`.

`NodeList titre e.getElementsByTagName('titre')` → récupère les titres de façon récursive. Il effectue une recherche complète dans le sous arbre. Il n'existe pas de fonction qui permet de récupérer la position dans l'arbre d'un « élément ».

`titre.item(0).getFirstChild().getNodeValue()` → je suis dans l'élément, je récupère son fils, pour récupérer la valeur du nœud.

3- Ajout d'un sous arbre / de nœuds *Cf. exemple slide 32.*

Il ne fait pas de contrôle après rajout. Il faudrait donc contrôler le schéma après un ajout.
Pour un rajout il faut rajouter toutes les balises.
Dans un premier temps il faut récupérer la racine et faire un
`racine.appendChild(d.createElement('film')) ;` → crée un fils à l'élément racine.
Ici on rajoute élément par élément mais il est possible de créer un sous arbre et de le rajouter d'un coup. Les modifications ici ne se font pas sur le disque mais sur le document qui est stocké est mémoire (d'où l'étape 5).

4- Affichage du document XML *Cf. exemple slide 33.*

Afficher c'est transformé. Transformer c'est prendre la représentation mémoire (sous forme d'objet) et de le transformer en flux d'octet en sortie.
Ici le résultat on le met sur `System.out` (flux d'octet).

5- Sauvegarde du document dans un fichier *Cf. exemple slide 33.*

La différence avec le point 4 est qu'on lui passe un objet `File` (fichier) en paramètre. La chaîne de caractère générée pendant le « `transform` » est reçue par l'objet `File`.

Tous les navigateurs intègrent DOM.

Tout ce qui est système ne se sérialise pas. Sérialiser : dumper la mémoire (photocopie de la mémoire avec des informations en plus).
Pour rappel, le but de XML est dans un premier temps de représenter et structurer les données et dans un second temps d'échanger les données. Si on fait de la sérialisation on ne peut pas échanger les données, ce qu'on ne souhaite pas. En effet Java sérialise alors que Python non (pickling).

JDOM

JDOM est une API simplifiée de DOM. JDOM ne respecte pas la spécification du W3C, seulement une partie.
La structuration reste un arbre.
Il est possible de créer des documents (depuis un document vierge ou un fichier), de sauvegarder, d'ajouter des éléments ou de supprimer.
JDOM part du principe que si on a une balise on a une valeur. Concernant les balises auto fermantes leur valeur est nulle.
Cf. exemple slide 37.