XML et Architectures Logicielles Distribuées

Baabda :
M. E. ABOU ZEID
M. E. MATTA

Zahlé :
Mlle. C. SAAD

Mejdelaya:
M. R. ABI NADER

PLAN DU COURS

- Chapitre 1: Le Langage XML
- **Chapitre 2: XPath et APIs XML**
- Chapitre 3: Les services web
- Chapitre 4: Les architectures distribuées
- Chapitre 5: Le Schéma XML

CHAPITRE 2

XPath et APIs XML

PLAN DU CHAPITRE

- APIs XML
- XPath
- XPath sous le Langage C#

APIs XML

APIs XML

- XML est uniquement **un langage de structuration** et de **représentation de données**.

- Pour réaliser des applications XML, on aura besoin d'un dispositif logiciel ("**interface**") permettant de manipuler le doc XML pour exploiter et modifier ses données
  - «*API XML*» ou «*Parseur XML*»
  - Écrits avec **différents langages de programmation**:
    - C#, Java, C/C++, etc.

APIs XML

- Un Parseur XML est un petit **module de programme** permettant de:
  - **Vérifier la validité et la conformité syntaxique du document XML**
  - **Manipuler le document XML**
    - Décomposer en éléments/attributs
    - Récupérer des données (accès), afficher, modifier, sauvegarder
- Plusieurs catégories de parseurs:
  - **Validant** vs. **non-validant** par rapport à la DTD/XSD
  - Supportant le modèle **DOM** (*Document Object Model*)
  - Supportant l'interface **SAX** (*Simple API for XML*)

APIs XML

Critères de comparaison	DOM	SAX
Efficacité mémoire		X
Traitement **en cours d'analyse** (manipulation au fur et à mesure de la lecture)		X
Orienté **événements** (« *event-based* »)		X
Accès **aléatoire** aux données	X	
Modification des données	X	
Contexte Maintenu (accès aux noeuds parents, enfants, etc.)	X	
Orienté **hiérarchie** («*tree-based*»)	X	
Orientée **objet**	X	
Chargement **complet** en **mémoire**	X	

XPath

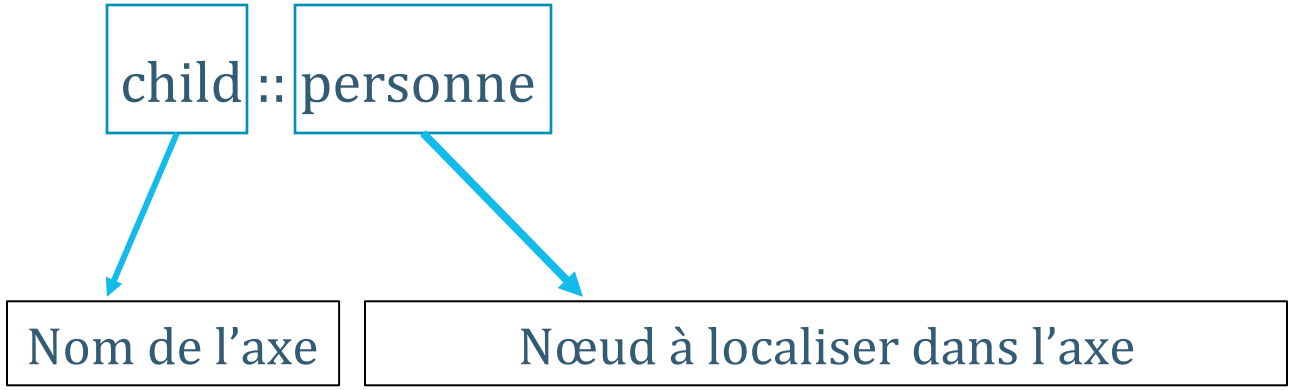
XPath

- Langage de requête, **non XML**.
- Peut être employé dans le contexte d'une autre technologie
  - XSL
  - XQuery
  - Etc.
- Permet de:
  - Sélectionner une partie d'un document XML
  - Filtrer les nœuds

- Formé d'un **chemin de localisation** composé de 3 parties:
  1. **Un axe**: direction dans l'arbre (relative ou absolue)
  2. **Un test de nœud**: identification des nœuds dans l'axe
  3. **0 à n prédicats** (optionnels)
     - des conditions (expressions booléennes) à respecter - sur chaque nœud trouvé
     - filtrer les nœuds sélectionnés par l'axe et le test de nœud

XPath

- Le **chemin de localisation**



=> Sélection de tous les éléments, fils du nœud courant, ayant pour nom « personne »

  - **N.B.:** en l'absence d'un axe, l'axe par défaut est ***child***

XPath

Les axes peuvent être (entre autres):

- **child**: les enfants directs du nœud contextuel
- **parent**: le parent du nœud contextuel, s'il y en a un.
- **descendant**: les descendants du nœud contextuel. Un descendant peut être un enfant, un petit-enfant...
- **ancestor**: contient les ancêtres du nœud contextuel. Cela comprend son père, le père de son père...
- **attribute**: contient les attributs du nœud contextuel
- **self**: contient seulement le nœud contextuel
- **following-sibling**: contient tous les nœuds frères du nœud contextuel.
- **preceding-sibling**: contient tous les prédécesseurs du nœud contextuel

XPath

Expression	Description
nom_du_noeud	Sélection de tous les nœuds ayant pour nom "*nom_du_noeud*"
/	Sélection du nœud racine
//	Sélection des nœuds dans le document, commençant par le nœud courant, qui correspondent la sélection, peu importe leur position
.	Sélection du nœud courant
..	Sélection du parent du nœud courant
@	Sélection d'un attribut
*	N'importe quel nœud élément
@*	N'importe quel nœud attribut
node()	N'importe quel type de nœuds (éléments/attributs)
text()	Les nœuds (dans le contexte courant) contenant du texte uniquement

XPath: Exemples

- **child::personne**
  - tous les éléments fils du nœud courant ayant pour nom *personne;*
- **child::***
  - tous les éléments fils du nœud courant;
- **descendant::***
  - tous les éléments descendants du nœud courant ;
- **attribute::titre**
  - l'attribut *titre* du nœud de contexte

- **self::carnet**
  - le nœud *carnet* courant ou un ensemble vide si cela ne correspond pas.
- **child::text()**
  - Tous les noeuds contenant du texte et qui sont fils du noeud courant
- **child::node()**
  - Tous les noeuds fils du noeud courant
- **child::personne/child::email**
  - rechercher l'ensemble des éléments *email* dans les nœuds *personne* du nœud de référence.
- **/child::carnet/descendant::email**
  - Rechercher l'ensemble des éléments *email* en partant de la racine *carnet.*

XPath: Exemples

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book>
        <title lang="en">Harry Potter</title>
        <price>29.99</price>
    </book>
    <book>
        <title lang="en">Learning XML</title>
        <price>39.95</price>
    </book>
</bookstore>
```

Document XML

- **//***
  - Tous les éléments du document
- **//book**
  - Tous les éléments book peu importe où ils sont dans le document
- **bookstore//book:**
  - Tous les éléments book descendants de bookstore peu importe où ils sont dans le document
- **/bookstore/***
  - Tous les éléments enfants de bookstore

XPath: Exemples

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book>
        <title lang="en">Harry Potter</title>
        <price>29.99</price>
    </book>
    <book>
        <title lang="en">Learning XML</title>
        <price>39.95</price>
    </book>
</bookstore>
```

Document XML

- **//title[@*]**
  - Tous les éléments title qui ont un attribute
- **/bookstore/book[last()]**
  - dernier book qui est enfant de bookstore
- **/bookstore/book[last()-1]**
  - avant dernier book qui est enfant de bookstore
- **/bookstore/book[position()<3]**
  - 2 premiers éléments book enfants de bookstore
- **/bookstore/book[price>35.00]/title**
  - Les éléments title dans book qui ont un élément price avec une valeur supérieure à 35.00

XPath: Fonctions et Opérateurs (entre autres)

Fonction	Description	Exemple
last	Determine le dernier nœud	**/descendant::personne[last()]** Retourne le dernier élément *personne*
position	Détermine le numéro du nœud de contexte courant	**/descendant::personne[position()=2]** Retourne la 2ème personne ⇔ **/descendant::personne[2]**
count	Retourne le nombre de nœuds de l'ensemble passé en argument	**count(/descendant::personne)** Retourne le nombre de *personne*
sum	additionne un ensemble de valeurs numériques	**sum(/descendant::personne/attribute::numero)** Resultat: 3
starts-with	détermine si la chaîne en 1e argument commence par le 2ème argument	**starts-with("ABC", "AB")** ➔ vrai **starts-with("ABC", "B")** ➔ faux
contains	détermine si la 1ère chaîne en argument contient le 2ème argument	**contains(string(/child::carnet) , "Dup")** ➔ vrai

XPath: Fonctions et Opérateurs (entre autres)

Fonction	Description	Exemple
not	Retourne Vrai pour une expression qui s'évalue comme fausse	**//personne [** **./prenom= 'John'** **and** **not (starts-with (@id, "INF"))** **]** Retourne les éléments personne qui: • Ont pour élément fils *prenom* 'John' • N'ont pas un attribut id commencant par 'INF'

Les opérateurs utilisables sont:

	+	-	*
=	!=		
<	<=	>	>=
and	or		
div	mod		

XPath sous le Langage C#

XPath sous le Langage C#

- **Soit le fichier XML suivant:**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<catalog>
  <cd id="1">
    <country>USA</country>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd id="2">
    <country>UK</country>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd id="3">
    <title>USA</title>
    <artist>Dolly Parton</artist>
    <price>10</price>
  </cd>
</catalog>
```

XPath sous le Langage C#

```
public static class Utilities
{
    static string fileName;
    static XmlTextReader reader;
    static XmlDocument doc;
    static XPathNavigator nav;
    static XPathExpression expr;
    static XPathNodeIterator iterator;

    //functions go here...

}
```

- **XmlDocument**: classe dans l'espace de noms *Windows.Data.Xml.Dom* qui consiste d'une représentation en mémoire d'un document XML. Elle est utilisée pour charger, valider, modifier, ajouter et positionner les données XML dans un document.
- **XPathNavigator**: classe définissant un modèle de curseur pour naviger et modifier les données XML en tant qu'instances de Xquery 1.0 et Xpath 2.0.
- **XPathExpression**: classe représentant une expression XPath compilée.
- **XPathNodeIterator**: classe qui fournit un itérateur sur une série de noeuds.

XPath sous le Langage C#

Méthodes de la classe XPathNavigator:

- CreateNavigator
- Clone
- Compile
- Evaluate
- SetValue

- ReplaceSelf
- DeleteSelf

- InsertAfter
- InsertBefore

- Select
- SelectSingleNode

- AppendChild

- MoveToChild
- MoveToFollowing
- MoveToParent

- GetAttribute
- CreateAttribute
- MoveToAttribute

- SelectChildren
- SelectAncestors
- SelectDescendants

***Note**: des exemples seront illustrés pour démontrer les méthodes soulignées*

XPath sous le Langage C#

- Chargement du document XML:

```
public static void LoadXmlDocument()
{
    try
    {
        fileName =  "..\\..\\Data\\catalog.xml";
        doc = new XmlDocument();
        nav = doc.CreateNavigator();
        reader = new XmlTextReader(fileName);
        doc.Load(reader);
        reader.Close();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Affichage des données de tous les éléments <cd>

```
public static void PrintAllElements()
{
    try
    {
        expr = nav.Compile("/catalog/cd");
        iterator = nav.Select(expr);

        while (iterator.MoveNext())
        {
            XPathNavigator nav2 = iterator.Current.Clone();
            Debug.WriteLine("ID: " + nav2.GetAttribute("id", "") + "\n"
            + "country: " + nav2.SelectSingleNode("country") + "\n"
            + "title: " + nav2.SelectSingleNode("title") + "\n"
            + "artist: " + nav2.SelectSingleNode("artist") + "\n"
            + "price: " + nav2.SelectSingleNode("price")
        );
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Affichage d'un CD selon son identifiant (selon la valeur de l'attribut id)

```
public static void PrintElementById(int id)
{
    try
    {
        expr = nav.Compile("/catalog/cd[@id='" + id.ToString() + "']");
        iterator = nav.Select(expr);
        iterator.MoveNext();
        XPathNavigator nav2 = iterator.Current.Clone();
        Debug.WriteLine("ID: " + nav2.GetAttribute("id", "") + "\n"
        + "country: " + nav2.SelectSingleNode("country") + "\n"
        + "title: " + nav2.SelectSingleNode("title") + "\n"
        + "artist: " + nav2.SelectSingleNode("artist") + "\n"
        + "price: " + nav2.SelectSingleNode("price"));
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Affichage du prix du dernier CD dans le document en utilisant la function **last()**:

```
public static void PrintLastElementPrice()
{
    try
    {
        expr = nav.Compile("//cd[last()]");
        iterator = nav.Select(expr);
        iterator.MoveNext();
        XPathNavigator nav2 = iterator.Current.Clone();
        Debug.WriteLine("price of last element: " + nav2.SelectSingleNode("price"));
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Utilisation des fonctions XPath **count()** et **sum()**:

```
public static void XpathFunctions()
{
    try
    {
        Debug.WriteLine(nav.Evaluate("count(/catalog/cd)").ToString());
        Debug.WriteLine(nav.Evaluate("count(/catalog/cd[country='UK'])").ToString());
        Debug.WriteLine(nav.Evaluate("count(/catalog/cd[price>'10' and country = 'USA'])").ToString());

        Debug.WriteLine(nav.Evaluate("sum(/catalog/cd[price>='10']/price)").ToString());
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Utilisation de la méthode **InsertAfter**:

```
public static void InsertAfterId(int id, string title, string artist, double price)
{
    try
    {
        int maxID = Convert.ToInt32(nav.Evaluate("count(/catalog/cd)")) + 1;

        expr = nav.Compile("/catalog/cd[@id='" + id.ToString() + "']");
        iterator = nav.Select(expr);
        iterator.MoveNext();
        XPathNavigator nav2 = iterator.Current.Clone();

        XmlReader newCd = XmlReader.Create(new StringReader(
                                    "<cd id='" + maxID.ToString() + "'>" +
                                    "<title>" + title + "</title>" +
                                    "<artist>" + artist + "</artist>" +
                                    "<price>" + price.ToString() + "</price>"
                                    + "</cd>")
                                );

        nav2.InsertAfter(newCd);
        doc.Save(fileName);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Utilisation de la méthode **SetValue**:

```
public static void AddTaxToPrices (double tax)
{
    try
    {
        double val, taxedVal;
        foreach (XPathNavigator n in nav.Select("//price"))
        {
            val = Convert.ToDouble(n.Value);
            taxedVal = val * (1+tax);
            n.SetValue(taxedVal.ToString());
        }
        doc.Save(fileName);
        PrintAllElements();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Remplacer un élément selon la valeur de son identifiant et persister les données:

```
public static void ReplaceElementById(int id, string title, string artist, double price)
{
    try
    {
        expr = nav.Compile("/catalog/cd[@id='" + id.ToString() + "']");
        iterator = nav.Select(expr);
        iterator.MoveNext();
        XPathNavigator nav2 = iterator.Current.Clone();
        nav2.InnerXml = "<title>" + title + "</title>" +
                        "<artist>" + artist + "</artist>" +
                        "<price>" + price.ToString() + "</price>";

        doc.Save(fileName);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

XPath sous le Langage C#

- Ajouter un élément et persister les données:

```
public static void AppendElement(string title, string artist, double price)
{
    try
    {
        int maxID = Convert.ToInt32(nav.Evaluate("count(/catalog/cd)")) + 1;
        nav.MoveToChild("catalog", "");
        XmlReader newCd = XmlReader.Create(new StringReader(
                                    "<cd id='" + maxID.ToString() + "'>" +
                                    "<title>" + title + "</title>" +
                                    "<artist>" + artist + "</artist>" +
                                    "<price>" + price.ToString() + "</price>"
                                    +"</cd>")
                                );

        nav.AppendChild(newCd);
        doc.Save(fileName);
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

Questions?