Université de Paris

Langages de balisage et leur validation : XML vs. JSON.

LENOIR Quentin - HADJAB Lynda

Année Universitaire 2021-2022

Introduction

La transmission de données textuelle est indispensable à bon nombre d'applications aujourd'hui et il existe diverses technologies pour les mettre en forme. Le JSON et le XML sont les principaux formats utilisés dans ce cadre, nous allons donc expliquer quelles sont leurs caractéristiques et dans quel cadre ils sont adaptés, en parlant dans une première partie du XML puis dans un deuxième temps du JSON pour finir avec une comparaison directe des deux langages.

I - XML :

1 - Qu'est-ce que le XML ?

Le langage de balisage extensible, abrégé XML est un méta-langage permettant de définir des langages à balises. Il est standardisé comme une recommandation par le W3C depuis 1998. Il décrit une classe d'objets de données appelés documents XML et décrit partiellement le comportement des programmes informatiques qui les traitent. Le XML est un profil d'application ou une forme restreinte du SGML, le langage standard de balisage généralisé [ISO 8879]. Par construction, les documents XML sont des documents SGML conformes.

2- Structure d'un document XML ?

Un document XML est structuré en trois parties : La première partie, appelée prologue, permet d'indiquer la version de la norme XML utilisée pour créer le document ainsi que le jeu de caractères utilisé dans le document (attribut facultatif). Ainsi le prologue est une ligne du type :

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

Ensuite la deuxième partie est une déclaration de type de document DTD, une grammaire permettant de vérifier la conformité du document XML, il s'agit d'un certain nombre de contraintes que doit respecter un document pour être *valide*. Ces contraintes spécifient quels sont les éléments qui peuvent apparaître dans le contenu d'un élément, l'ordre éventuel de ces éléments et la présence de texte brut. Elles définissent aussi, pour chaque élément, les attributs autorisés et les attributs obligatoires.

Les déclarations de type de documents DTDs utilisent un langage spécifique non XML pour définir les règles structurelles. Un fichier de DTD peut contenir principalement deux types de déclarations: des déclarations d'éléments, indiquent les éléments pouvant être inclus dans un document et l'organisation du contenu de chaque élément (éléments fils ou texte). Et des déclarations d'attributs, définissent les attributs pouvant être associés à un élément ainsi que leur type.

Un exemple d'une DTD, elle peut être définie soit à l'intérieur d'un document XML soit dans un fichier à part avec comme extension pour le fichier ***.dtd*** .

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE message [
        <!ELEMENT message (expediteur, destinataire, texte)>
        <!ELEMENT expediteur (identite)>
        <!ELEMENT destinataire (identite)>
        <!ELEMENT identite (prenom, nom, email)>
        <!ELEMENT nom (#PCDATA)>
        <!ELEMENT prenom (#PCDATA)>
        <!ELEMENT email ANY>
        <!ELEMENT texte ANY>
        <!ATTLIST nom
          age CDATA #IMPLIED
          sexe (m|f) #REQUIRED
        >
      ]>
```

Et enfin la dernière composante d'un fichier XML est l'arbre des éléments, une hiérarchie de balise comportant des attributs, tels que un attribut est une paire clé valeur écrite sous la forme Clé="value".

```
<message>
  <expediteur>
            <identite>
 <prenom>Maxime</prenom>
 <nom age="23" sexe="m">Charpentier</nom>
 <email>maximeCha@etudiant.univ.fr</email>
            </identite>
  </expediteur>
  <destinataire>
     <identite>
               <prenom>Bob</prenom>
               <nom age="30" sexe="m">Dupont</nom>
               <email>bobdupont@provider.com</email>
            </identite>
         </destinataire>
          <message>mon message</message>
    </message>
```

Avec un fichier XML on a la possibilité d'ajouter des commentaires, ils peuvent être insérés n'importe où dans le document avec la syntaxe : <!-- Ceci est un commentaire -->

3- La validation des documents XML :

Les navigateurs comme Internet Explorer contiennent des analyseurs XML intégrés, qui vérifient si le document est bien formé, et s'il valide l'analyseur. Un document est bien formé s'il suit les règles syntaxiques de base du XML et il est valide s'il vérifie les règles mentionnées par la DTD pour le XML.

Il existe deux types de validations de DTD : la validation interne et la validation externe. Pour les validations internes, nous écrirons la DTD entière dans le même fichier que le fichier XML, qui peut être utilisé pour la validation. De même, la validation externe validera le XML sur la base de la DTD écrite dans un fichier séparé avec l'extension .dtd.

Exemple de document XML : Le document suivant est-il valide ?

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
     <!DOCTYPE address [
         <!ELEMENT address (name,company,phone)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT company (#PCDATA)>
        <!ELEMENT phone (#PCDATA)>
        <!ELEMENT zipCode (#PCDATA)>
]>

<address>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</address>
```

L'exemple ci-dessus est dit bien formé car il définit bien le type de document. Ici, le type de document est le type d' élément, et il comprend un élément racine nommé address, chacun des éléments enfants parmi le name, la company et le phone est entouré d'une balise explicite et enfin l'ordre de balise est conservé, par contre le document n'est pas valide car pour la balise address il manque la balise zipCode (balise enfant) et donc le DTD va pas considérer le document comme valide.

Pour conclure, XML est très simple, il utilise très peu de syntaxe et de règles lors du codage. Il ne nécessite aucun outil ou logiciel spécial pour écrire le code, il peut être écrit par le biais d'un simple éditeur de code ou d'un bloc-notes. Vous pouvez créer des balises vous-même, il n'est donc pas nécessaire de se souvenir d'une balise quelconque. Cela rend le partage des données très facile. Et il est indépendant de la plate-forme et du langage de programmation, il peut donc être utilisé sur n'importe quel système et supporte les changements technologiques lorsqu'ils se produisent, et il garantit que le document XML est exempt de toute erreur de syntaxe grâce à la validation qui est faite avec DTD.

II - JSON

1. Définition du JSON

Le JSON est un acronyme pour Javascript Object Notation, puisque comme son nom l'indique il est dérivé de la notation des objets en Javascript. C'est un format de données textuelles créé par Douglas Crockford entre 2002 et 2005, dont la première norme ECMA a été publiée en 2003. Le JSON est composé de deux structures, dérivées du JS : les objets qui associent des clés à des valeurs (exemple ci-dessous), et des tableaux. Ces structures peuvent contenir 4 types de données : des chaînes de caractères, des nombres, des booléens et *null*.

{ clé : "valeur", clé2 : "valeur2" } ["1",-1.5,{prénom : "Quentin"}]

Nous pouvons étudier un exemple d'un JSON exprimant les caractéristiques d'un chien. On constate que les clés sont elles aussi sous forme de chaînes, il n'est pas valide d'avoir une clé qui n'est pas une chaîne de caractère, contrairement au JavaScript. Le JSON n'a aucun problème pour contenir des objets qui peuvent être complexes ou non, de même pour des tableaux. En utilisant la méthode JSON.parse() disponible nativement avec le Javascript on obtient les données souhaitées structurées de la même façon à la différence que les clés ne sont pas des chaînes de caractères.

```
{
  "Espèce": "Chien",
  "Race": "Labrador Retriever",
  "Age": 6,
  "Caractéristiques": {
        "couleurYeux": "Marron",
        "couleurPoil": "Jaune",
        "poids": "62"
  }
}
```

-> JSON.parse() ->

```
Age: 6
▼ Caractéristiques:
    couleurPoil: "Jaune"
    couleurYeux: "Marron"
    poids: "62"
  ▶ [[Prototype]]: Object
Espèce: "Chien"
Race: "Labrador Retriever"
```

Le JSON possède comme désavantage, de ne pas être très lisible pour un humain, en outre, il n'accepte pas les commentaires. Tout caractère inattendu lorsque l'on essaye de parser un JSON lèvera une exception. C'est un choix délibéré du créateur de JSON, qui voulait éviter les problèmes d'interopérabilité et garder le format le plus minimaliste possible.

Il y a cependant des solutions pour contourner ça, nous avons choisi de montrer HumanJSON, qui surcharge le format JSON en permettant d'insérer des commentaires, ou par exemple permettre d'utiliser des clé qui ne sont pas des chaînes de caractères, pour que cela soit plus lisible par un humain. Voici deux exemples :

```
{
  JSON: "a string"

  Hjson: a string

  # notice, no escape necessary:
  RegEx: \s+
}
```

```
{
  # hash style comments
  # (because it's just one character)

  // line style comments
  // (because it's like C/JavaScript/...)

  /* block style comments because
     it allows you to comment out a block */

  # Everything you do in comments,
  # stays in comments ;-}
}
```

Il existe de nombreuses autres bibliothèques pour JSON, que ce soit pour permettre sa lecture sur d'autres langages comme Python ou C++, ou pour standardiser certains types de données comme de la géométrie ou de la topologie. On peut aussi évoquer le BSON, signifiant Binary JSON, qui est utilisé par les bases de données MongoDB afin de stocker des données en binaire.

2. Validation d'un JSON

La validation d'une structure JSON est réalisée par le parseur qui va se charger de le lire, cependant il ne contrôle pas la structure des données en elle-même, ainsi si on s'attendait à recevoir un certain type de données et qu'on en reçoit un autre cela peut causer un problème. C'est pourquoi on peut utiliser des schémas pour valider les données en JSON. A l'aide de bibliothèques on peut définir des schémas de validation, qui sont en fait eux mêmes des JSON avec une certaine structure, qui seront ensuite utilisés pour valider la structure du JSON contenant les données, en voici un exemple :

```
{
        "properties": {
            "productId": {
    "description": "The unique identifier for a product",
    "type": "integer"
},
            "productName": {
"description": "Name of the product",
 "type": "string"
            },
            "price": {
    "description": "The price of the product",
    "type": "number",
    "exclusiveMinimum": 0
            }
        },
        "required": [ "productId", "productName", "price" ]
}
```

Grâce à l'objet contenu dans la clé properties, on voit quelles clés sont attendues ainsi que le type. La description est uniquement à but indicatif pour un humain qui chercherait à lire le schéma de validation. On peut aussi spécifier des contraintes, par exemple sur les nombres on peut définir des bornes minimales et maximales, exclusives ou non.

3. Conversion JSON / XML

Afin de faciliter la conversion XML vers du JSON et inversement, il a été introduit comme langage de balisage le JsonML, qui permet d'éviter d'avoir un parseur XML pour une application recevant des données XML et qui les traite en JSON par exemple. Sa structure est simple : les tableaux symbolisent les nœuds d'un XML, les objets sont les attributs de ce nœud, et les chaînes de caractères sont son contenu. Voici un exemple de JsonML, où le noeud "person" figure en haut, suivi d'un objet qui contient donc ses attributs "created" et "modified", pour ensuite avoir des tableaux symbolisant ses enfants ("firstName","lastName",etc). On retrouve donc bien la bonne structure XML, et ce sans la connaître au préalable.

```
{"person",
  {"created":"2006-11-11T19:23",
   "modified":"2006-12-31T23:59"
  },
  ["firstName", "Robert"],
  ["lastName", "Smith"],
  ["address", {"type":"home"},
    ["street", "12345 Sixth Ave"],
    ["city", "Anytown"],
    ["state", "CA"],
    ["postalCode", "98765-4321"]
  ]
}
```

```
<!-- XML representation of a person record -->
<person created="2006-11-11T19:23"
modified="2006-12-31T23:59">
    <firstName>Robert</firstName>
    <lastName>Smith</lastName>
    <address type="home">
        <street>12345 Sixth Ave</street>
        <city>Anytown</city>
        <state>CA</state>
        <postalCode>98765-4321</postalCode>
    </address>
</person>
```

Pour conclure, le JSON a comme avantage d'être minimaliste, en se limitant à une structure simple et des données limitées et définies, il est ainsi facile à parser en utilisant du Javascript ou bien des bibliothèques très bien maintenues qui sont conçues pour d'autres langages, ce qui en fait un format très portable et adapté pour le transport de données. Il permet aussi de traiter des résultats de requêtes AJAX efficacement, contrairement à du XML qui nécessite un parseur. Le JSON possède des désavantages, car en plus de ne pas pouvoir faire de commentaires, il n'existe pas de namespace contrairement au XML , et limite donc un peu l'extensibilité. Un humain pourrait avoir des difficultés à le lire s' il est trop complexe sans avoir un outil approprié pour l'afficher correctement, là où pour le coup le XML est plus lisible.

Conclusion

Nous avons présenté les différentes caractéristiques propres au XML et JSON, on peut donc constater maintenant les différences, on peut citer l'encodage, le XML en supporte plusieurs alors qu'en revanche le JSON ne supporte que l'UTF-8. Nous avons déjà parlé des différences de lisibilité, on peut noter que les tableaux ne sont supportés qu'en JSON et non en XML, mais le XML est plus sécurisé que le JSON, car si ce n'est pas surveillé le JSON parsé peut exécuter du code malveillant introduit dans les données. Il existe aussi des vulnérabilités XML mais elles sont beaucoup plus simples à gérer. Pour conclure, le JSON est bien plus adapté pour gérer des données en web, et est portable facilement ailleurs, mais le XML offre une bien meilleure extensibilité et personnalisation pour les structures.

Bibliographie

https://fr.wikipedia.org/wiki/Langage_de_balisage

https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

https://json-schema.org/

https://fr.wikipedia.org/wiki/Extensible_Markup_Language

https://www.w3schools.com/js/js_json_xml.asp

https://www.geeksforgeeks.org/difference-between-json-and-xml/

https://www.xul.fr/xml.php