# XML

## הגדרה

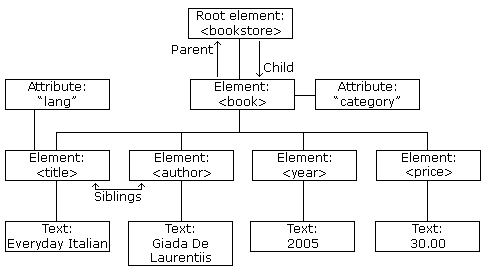
XML (eXtensible Markup Language) הוא תקן לקידוד וייצוג נתונים באופן טקסטואלי. ה-XML שימושי כמעט בכל תחום של מחשבים. חשיבותו הגדולה ביותר היא בחיבורים בין מערכות שונות. לדוגמה: קיים תקן כיצד לייצג מידע על חולה והטיפולים שהוא עובר ("תיק חולה") באמצעות מסמך XML. כך יכולה המערכת של בית החולים לשלוח את המידע הזה לרופא המשפחה שיכול לקלוט לתוך מערכת המידע שלו את התרופות אותם קיבלו חוליו במהלך אשפוזם. כדי לתמוך בהחלפת מידע מסוג זה פותחו תקנים להעברת מסמכי XML כדוגמת SOAP. היתרון של XML הוא שהמידע קריא ונוח למשתמש, ולכן משמש כדי לאחסן מידע הנועד גם לקריאה. אמנם הוא לא מיועד להעביר כמויות גדולות של מידע, לשם כך משתמשים במסדי נתונים.

תקן ה-XML לא מגדיר איזה מידע יוצג אלא מגדיר כיצד לייצג מידע באופן כללי. הייצוג עצמו מתבצע תוך שימוש בתגיות (Tags) שמגדירות צמתים. כל צומת יכולה להיות רכיב מידע (Element) בעצמה או להכיל מספר רכיבי מידע.

מבחינה תחבירית XML דומה מאוד ל-HTML, אמנם ההבדל המרכזי ביניהם הוא ש-XML, בניגוד ל-HTML, משמש רק לאחסון המידע ולא לבצע עליו פעולות. מסיבה זו XML איננה מוגדרת כשפת תכנות. גם מבחינה תחבירית יש ביניהם מספר הבדלים.

## תחביר

הייצוג של המידע מתבצע באמצעות תגיות המגדירות אלמנטים. בתוך אלמנט ניתן לאחסן את המידע עצמו או אלמנטים נוספים. לכל אלמנט יש הצהרה פותחת והצהרה סוגרת. בנוסף, לכל אלמנט אפשר להוסיף תכונות (Attributes) כדי לארגן את המידע ולהקל על החיפוש. ב-XML טבעי מאוד לסדר מידע באופן היררכי (עץ). לדוגמה, קובץ XML המאחסן מידע על חנות ספרים:

<?xml version="1.0" encoding="UTF-8**"**?>  
<bookstore>  
   <book category="cooking">  
      <title lang="en">Everyday Italian</title>  
      <author>Giada De Laurentiis</author>  
      <year>2005</year>  
      <price>30.00</price>  
   </book>  
   <book category="children">  
      <title lang="en">Harry Potter</title>  
      <author>J K. Rowling</author>  
      <year>2005</year>  
      <price>29.99</price>  
   </book>  
</bookstore>

קובץ XML המוגדר לפי כל הכללים מכונה "well formed". ההזחות אינן נצרכות למערכת המפענחת את המידע, אך הן כן חשובות כדי שהמידע יהיה קריא.

לכל קובץ XML חייב להיות אלמנט אחד ויחיד המכונה שורש. הקובץ צריך להתחיל ולהסתיים עם התגית המייצגת אלמנט זה. בתוך אלמנט השורש, בדרך כלל נגדיר מספר אלמנטים עם שם זהה, אך יכולים להיות עם תכונות שונות. כל אלמנט הוא עצמאי ואינו תלוי באחרים, לכן יכול להכיל שדות אחרים מהאלמנטים האחרים.

כאמור לעיל, XML לא מגדיר איזה מידע יוצג וגם לא דורש התאמה בין האלמנטים, אולם ניתן להגדיר את המבנה של קובץ XML על ידי שימוש בקובץ נוסף, שמגדיר באילו תגיות ניתן להשתמש, מה המבנה שלהן ואיזה מידע יכול להיות מאוחסן בכל אחת מהן. בעבר השתמשו בפורמט DTD כדי להגדיר את התחביר. כיום משתמשים בפורמט XSD (המכונה גם Schema), שבעצמו מנוסח ב-XML (לעומת DTD שיש לו תחביר נפרד).

שמות האלמנטים המיוצג בתגיות הוא case-sensitive, כלומר רגיש לאותיות קטנות וגדולות. ניתן להגדיר אלמנט ריק אם שמים את הלוכסן לאחר שם האלמנט. לדוגמה, <book/>.

## ישויות

ישנם מספר תווים שלא נוכל להשתמש בהם בצורה הפשוטה שלהם בקובץ XML, שכן יש להם משמעות אחרת. לכל תו כזה מוגדרת ישות שבאמצעותה ניתן לייצג את התו. נפרט על חמישה ישויות:

" - &quat

& - &amp

' - &apos

> - &lt

> - &gt

## שימוש ב-Java

ב-Java מוגדרת ספרייה בשם org.w3c.dom שבאמצעותה נוכל לקרוא את המידע השמור בקובצי XML. השימוש בספרייה נעשה בכמה שלבים:

1. ליצור אובייקט מסוג Document המייצג קובץ XML. הבנייה שלו נעשית באמצעות אובייקט מסוג DocumentBuilder.
2. ליצור אובייקט מסוג NodeList המכיל רשימה של אלמנטים. כדי להכניס אליו אלמנטים יש להשתמש בשיטת getDocumentElement על האובייקט מסוג Document. הברירת מחדל היא להכניס את כל האלמנטים בקובץ, אמנם נוכל לבחור אלמנטים לפי שם.
3. כיוון שיכול להיות מספר אלמנטים עם אותו שם, NodeList אינו בהכרח מכיל אלמנט יחיד אלא רשימה של אלמנטים. כדי לקבל אלמנט כלשהו ניצור אובייקט Node שיכיל אלמנט בודד. נכניס אל Node אלמנט באמצעות שיטת item על האובייקט NodeList המקבלת את מספר האלמנט ברשימה. ניתן לעשות זאת באמצעות לולאה העוברת על כל האלמנטים ב-NodeList.
4. כל אלמנט יכול להכיל מספר אלמנטים, לכן שוב ניצור NodeList שיקבל את כל האלמנטים הפנימיים של אלמנט נוכחי, באמצעות שיטת getChildNodes על האובייקט מסוג Node. כל אלמנט פנימי נכניס אל Node פנימי, כמו שתיארנו בשלב 3.
5. המעבר על האלמנטים שהם עצמם מכילים מידע בדרך כלל יעשה באמצעות תנאי switch, כי לא כל אלמנט חיצוני מכיל את אותם אלמנטים פנימיים.

לדוגמה, בקובץ XML (מימין) המייצג רשימה של סטודנטים, נוכל להעביר את כל המידע שנמצא בקובץ אל רשימה מקושרת של אובייקטים המייצגים סטודנט, באמצעות הקוד משמאל:

<?xml version="1.0"?>

<University>

<Student degree="PhD">

<FirstName>Chaya</FirstName>

<LastName>Glass</LastName>

<id>111</id>

<age>21</age>

</Student>

<Student>

<FirstName>Tal</FirstName>

<LastName>Negev</LastName>

<id>222</id>

</Student>

</University>

File inputFile = new File("student.xml");  
DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
DocumentBuilder builder = factory.newDocumentBuilder();  
Document doc = builder.parse(inputFile);

NodeList nodeList = doc.getDocumentElement().getElementsByTagName("Student");

for (int studentIdx = 0; studentIdx < nodeList.getLength(); studentIdx++) {  
 Node studentNode = nodeList.item(studentIdx);  
 if (studentNode.getNodeType() == Node.*ELEMENT\_NODE*) {

Student student = new Student();  
 studentList.add(student);

NodeList studentAllNodes = studentNode.getChildNodes();  
 for (int stIdx = 0; stIdx < studentAllNodes.getLength(); stIdx++){  
 Node stInnerNode = studentAllNodes.item(stIdx);

switch (stInnerNode.getNodeName()) {  
 case "FirstName": student.firstName = stInnerNode.getTextContent(); break;  
 case "LastName": student.lastName = stInnerNode.getTextContent(); break;  
 case "id": student.id = Integer.*parseInt*(stInnerNode.getTextContent()); break;  
 case "age": student.age = Integer.*parseInt*(stInnerNode.getTextContent()); break;

} } } }

## XPath

זהו תחביר שבאמצעותו ניתן לגשת ישירות לאיזושהי קבוצה או איבר בודד של אלמנט או ערך, בלי לעבור את כל התהליך שתיארנו בסעיף קודם. לפי הגדרת XPath כדי לגשת לאלמנט כלשהו יש לרשום את כל האלמנטים בהיררכיה עד אליו כאשר בין כל רמת היררכיה מפריד לוכסן "/". תחביר זה דומה ל-path של קובץ המערכת הקבצים במחשב, אלא שניתן להוסיף לו כל מיני אופרטורים ותנאים כדי לקבל קבוצה ספציפית של אלמנטים או ערכים.

### תנאים

ב-XML יכולים להיות בכל רמת היררכיה מספר אלמנטים עם תגית זהה, לכן אם נציין תגית זו ב-XPath, הוא יכנס רקורסיבית לכל אחד מאלמנטים אלו. במקרה זה, נוכל לגרום לכך שיכנס רק לאלמנטים המקיימים תנאי מסוים. נעשה זאת באמצעות פתיחת סוגריים מרובעות לאחר שם התגית, ובהם נציין את התנאי. התנאי יכול להיות על האלמנטים הפנימיים או על המיקום של האלמנט בו אנו מעוניינים. לאחר התנאי נוכל להמשיך עם החיפוש בהיררכיה. נביא מספר דוגמאות:

/bookstore/book[1] – select the first book

/bookstore/book[last()-1] – select the last but one book

/bookstore/book[position()<3] – select the first 2 books

/bookstore/book[price>35.00]/title – select all the titles of books that have a price greater than 35

### תכונות

כדי לבחור תכונות יש להשתמש @ ולאחריו שם התכונה שאנו מחפשים. לדוגמה,

/University/Student[1]/@degree -- gets the degree attribute of student 1

### אופרטורים נוספים

* כדי לשלב תוצאות משני XPath שונים, נשים ביניהם אופרטור "|".

University/Student/FirstName **|** /University/Student/Address

* כדי לבחור את כל האלמנטים של צומת כלשהי נרשום לאחריה כוכבית "\*".

/University/Student/\*

* אם רוצים את כל האלמנטים עם תגית מסוימת, שיכול להיות שיש להם היררכיה שונה, נשתמש באופרטור פעמיים לוכסן "//" על תגית זו בלי לציין את ההיררכיה שלפניה.

University//age

* כדי לחזור שלב אחורה בהיררכיה נשתמש באופרטור שתי נקודות "..".

/University/Student/Address[Zip=40792]/../FirstName

## XQuery

זהו תחביר המיועד לבצע שאילתה על קובץ XML. XQuery מזכיר את שפת SQL על מסדי נתונים. משתמשים ב-XPath כדי לגשת לערך או אלמנט ספציפי. נשמור את הערך או האלמנט שהוחזר מה-XPath במשתנה שנגדיר אותו באמצעות אופרטור $. ישנם 5 מילים שמורות המרכיבות שאילתת XQuery, ראשי תיבות שלהם הוא FLWOR.

1. for - בוחר קבוצה של אלמנטים. בדרך כלל באמצעות XPath.
2. let - קושר קבוצת אלמנטים אל משתנה. אופציונאלי.
3. where - מסנן אלמנטים מתוך הקבוצה.
4. order by - ממיין את האלמנטים. אופציונאלי.
5. return - איזה ערך להחזיר מהשאילתה.

דוגמאות:

* for $x in /University/Student

where $x/id > 0

return $x

* for $x in /University/Student

let $avg\_age := avg(/University/Student/age)

where $x/age < $avg\_age + 1

return $x/id

## Validation

XML לא מגדיר איזה מידע יוצג וגם לא דורש התאמה בין האלמנטים. אולם ניתן להגדיר את המבנה של קובץ XML על ידי שימוש בקובץ נוסף, שמגדיר באילו תגיות ניתן להשתמש, מה המבנה והסדר שלהן, ואיזה מידע יכול להיות מאוחסן בכל אחת מהן. יש שני שיטות לעשות זאת DTD ו-XSD (המכונה גם Schema). DTD (Document Type Definition), בניגוד ל-XSD, היא שפה בפני עצמה ולכן כמעט ולא בשימוש. נתרכז בשיטת XSD.

### XSD

XSD (XML Schema Definition) הוא תקן המשמש להגדרת מבנה של קבצים מסוג XML.‏ היתרון בשיטה זו הוא ש-XSD עצמו מהווה קובץ XML.

בתחילת קובץ XSD יש להצהיר מאיפה מייבאים את מרחב השם שבו כל ההצהרות שנשתמש בהן. מרחב שם זה הוא xs, לכן כל ההצהרות יתחילו בו. ב-XSD קיימים שני סוגי אלמנטים: פשוט (Simple Element) ומורכב (Complex Element). אלמנט פשוט מכיל אך ורק אוסף של תכונות (attributes), בעוד שאלמנט מורכב יכול להיות רצף של אלמנטים פנימיים שונים. כל אלמנט פשוט או מורכב נוכל להגדירו כרצף (sequence). המשמעות היא שסדר הערכים הפנימיים חשוב, כך שאם בקובץ XML הסדר לא יהיה תואם בכל אלמנט - הקובץ יוגדר לא תקין. לכל אלמנט פשוט נצטרך להגדיר את סוג הנתונים אותו מכיל. נוכל גם ליצור הגדרה של סוג ולהשתמש בו. לכל אלמנט מורכב נוכל להגדיר את מספר המופעים שלו בקובץ באמצעות minOccurs ו-maxOccurs. כאשר לא מוגדרים הם מאותחלים אוטומטית ל-1, כלומר לכל אלמנט מורכב יהיה מופע אחד בלבד.

דוגמה לסכמת XSD בסיסית להגדרה של מדינה.

**<xs:schema** xmlns:xs="http://www.w3.org/2001/XMLSchema"**>**

**<xs:element** name="country" type="Country"**/>**

**<xs:complexType** name="Country"**>**

**<xs:sequence>**

**<xs:element** name="name" type="xs:string"**/>**

**<xs:element** name="population" type="xs:decimal"**/>**

**</xs:sequence>**

**</xs:complexType>**

**</xs:schema>**

# JSON

## הגדרה

JSON (ראשי תיבות של JavaScript Object Notation) הוא פורמט טקסטואלי קל להבנה, המיועד להעברת מידע בצורה של מפתח-ערך (כמו טיפוסים פשוטים, מערכים, ואובייקטים), באמצעות מחרוזות טקסט פשוטות. קבצים בפורמט זה הם קבצי טקסט פשוטים בעלי סיומת '.json' . אף על פי שפורמט זה פותח בתחילה לשימוש ב-JavaScript ונעשה בו שימוש נרחב בשפה זו, JSON הוא מבנה נתונים לא תלוי שפה, ותמיכה מוכנה בהמרה לפורמט JSON קיימת במגוון שפות תכנות, גם ב-Java.

השימוש העיקרי של הפורמט הוא להעברת אובייקטים בין שרת לצרכן כתחליף לפורמט XML הכבד והמסובך. הרעיון מאחורי פורמט JSON הוא שלוקחים אובייקט מתוכנית אחת הכתובה בשפה כלשהי, ממירים אותו למחרוזת טקסט פשוטה בפורמט מוכר, שולחים אותו לתוכנית אחרת שיכולה להיות כתובה בשפה אחרת, ולבסוף ממירים את המחרוזת חזרה לאובייקט המקורי באופן המותאם לשפה הנוכחית. ממירים דווקא למחרוזת כדי שכל שפה תדע להתמודד איתו, שהרי בכל שפה מטפלים במחרוזות.

## תחביר

קובץ JSON מתחיל ונגמר בסוגריים מסולסלות. לאחר כל פרמטר יהיה פסיק.

* **טיפוסים פשוטים** - רושמים את שם המשתנה בגרשיים, לאחריו נקודותיים ':', ואז את ערך המשתנה: מספר, ערך בוליאני (true או false), מחרוזת (בגרשיים), או null.
* **מערכים** - לאחר הנקודותיים פותחים סוגרים מרובעים, ובהם ערכי המערך שביניהם פסיק.
* **אובייקטים** - לאחר הנקודותיים פותחים סוגריים מסולסלות, ומכניסים את שדות האובייקט - שם וערך כמו שתיארנו.

דוגמא לקובץ JSON המכיל אובייקט אדם, עם שדות טיפוסים פשוטים, אובייקט פנימי address, ומערך המכיל אובייקטים של מספרי טלפון:

{

"firstName": "John",

"isAlive": true,

"age": 27.5,

"address": {

"streetAddress": "21 2nd Street",

"city": "New York",

"state": "NY",

},

"phoneNumbers": [

{

"type": "home",

"number": "212 555-1234"

},

{

"type": "mobile",

"number": "123 456-7890"

}

],

}

## Json-simple

json-simple jar היא ספריית Java פשוטה המאפשרת לנתח, ליצור, להפוך ולעבד קבצי JSON. היא אינה מוגדרת ב-Java אלא יש להוריד ולייבא אותה. ספרייה זו מורכבת מ-13 מחלקות, המרכזיות שבהן הן:

1. **JSONParser** - נועדה לניתוח מחרוזת של JSON, הבנאי מקבל מחרוזות String או אובייקט זרימת תווים Reader.
2. **JSONObject** - אובייקט המייצג אובייקט שחולץ מ-JSON. הבנאי מקבלת מחרוזת JSON שמייצגת את האובייקט. קיימות בו שיטות המאפשרות לחלץ את כל המחרוזות JSON הקיימות באובייקט זה, לפי הטיפוס שלהם. בנוסף, מחלקה זו יורשת ממבנה הנתונים Map, ולכן על אובייקט זה ניתן להפעיל את כל השיטות שיש ב-Map.

יש לו גם שיטת writeJSONString(Map map, Writer out) המקבלת אובייקט Map ואובייקט זרימת תווים. באמצעות שיטה זו ניתן להמיר אובייקט Map למחרוזת JSON.

1. **JSONArray** - אובייקט המייצג מערך שחולץ מ-JSON. הבנאי מקבל מחרוזת JSON שמייצגת את המערך. קיימות בו שיטות המאפשרות לחלץ את כל המחרוזות JSON הקיימות באובייקט זה, לפי הטיפוס שלהם. בנוסף, מחלקה זו יורשת ממבנה הנתונים List, ולכן על אובייקט זה ניתן להפעיל את כל השיטות שיש ב-List.

יש לו גם שיטת writeJSONString(List list, Writer out) המקבלת אובייקט List ואובייקט זרימת תווים. באמצעות שיטה זו ניתן להמיר אובייקט List למחרוזת JSON.

1. **JSONValue** - נועדה להמרת מחרוזות JSON לאובייקטים של Java, וכן ההיפך.

JSONStringWrite(Object obj, Writer out) המשמשת ליצירת מחרוזת JSON מאובייקט Java.

Parse(Reader in) או parse(String s) מקבלות מחרוזת JSON ומחזירה אובייקט מסוג Object עליו ניתן לעשות casting. מומלץ להשתמש דווקא בשיטת parseWithException, העושה פעולה דומה רק שאינה מתעלמת מחריגות.

ה-API הבסיסי ביותר של המחלקות בקישור הבא:

<http://alex-public-doc.s3.amazonaws.com/json_simple-1.1/index.html>