

# XML

Пугачёв Константин (K.V.Pugachev@inp.nsk.su)

2020-10-05

# Описание данных в XML

# XML: eXtensible Markup Language

- Произошёл от SGML (1969) в ~1998
- Для компьютеров и людей
- Для Интернет-документов
- Иерархический
- Расширяемый

# XML-документ

```
<!-- ЗАГОЛОВОК -->
<?xml version="1.0" standalone="yes" encoding="UTF-8"?>

<!-- ТИП ДОКУМЕНТА -->
<!DOCTYPE mydoc SYSTEM "mydefinition.dtd">

<mydoc> <!-- корневой элемент, иерархия -->
  <tag attr="attrval">
    <anothertag anotherattr="value" />
    <![CDATA[ сырые данные ]]>
  </tag>
</mydoc>
```

# XML и мир

- XML - описывает данные
- HTML, XHTML - братья по маrazуму
- DTD, XSD - описывают метаданные (структуру)
- XSLT - осуществляет преобразование
- XPath - осуществляет поиск/матчинг
- JAXB, XmlSerializer - сериализация/десериализация объектов в XML и обратно

# Буковки-значки

- <, >, /, =, " - исп. для тегов и атрибутов
  - `<a href="https://example.com">click me</a>`
- &что-то; - описание символа в терминах XML
  - `&lt;` = `<`
  - `&gt;` = `>`
  - `&quot;` = `"`
  - `&amp;` = `&`
  - `&#x20;` = символ с кодом 0x20
  - `&#32;` = символ с кодом 32
- `<![CDATA[ сырые данные ]]>`

# XML: универсальность и расширяемость

- XML: теги, иерархия, пространства имён

```
<ns:tag attr="attrval">  
  <ns:anothertag anotherattr="value" />  
</ns:tag>
```

- Пользователь: грамматика для конкретного применения, валидация
  - У `ns:tag` есть атрибуты `x`, `attr`, причём `attr` обязателен
  - У `ns:tag` может быть 1 и более детей `ns:anothertag`
  - У `ns:anothertag` есть атрибуты `y`, `anotherattr` причём `anotherattr` обязателен
  - У `ns:anothertag` строго ноль детей

# Валидация XML



# Схема документа (DTD)

doc.xml:

```
<?xml version="1.0"?>
<!DOCTYPE mydoc SYSTEM "mydoc.dtd">
<mydoc>
  <tag attr="attrval">
    <anothertag anotherattr="value" v="1" />
  </tag>
  <tag2 a="1" />
  <tag2 a="2" />
</mydoc>
```

mydoc.dtd:

```
<!ELEMENT mydoc (tag, tag2+, tag3?)>
<!ELEMENT tag (anothertag)>
<!ATTLIST tag anotherattr CDATA #REQUIRED
            v CDATA #FIXED "1"
            x CDATA #IMPLIED
>
<!ELEMENT tag2 EMPTY>
<!ATTLIST tag a (1|2|3) "1">
```

# Схема документа (XSD)

mydoc.xsd:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="tagt">
    <xs:attribute name="attr" type="xs:string" use="required" />
    <xs:attribute name="v" type="xs:string" use="required" />
    <xs:sequence>
      <xs:element name="anothertag">
        <xs:complexType>
          <xs:attribute name="anotherattr" type="xs:string" />
          <xs:attribute name="v" type="xs:string" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="tag2t">
    <xs:attribute name="a" type="xs:string" use="required" />
  </xs:complexType>

  <xs:element name="mydoc">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="tag" type="tagt" />
        <xs:element name="tag2" type="tag2t" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Пример валидации в Node.js: libxml-xsd

```
npm install libxml-xsd
```

```
var xsd = require('libxml-xsd'), fs = require('fs');  
  
// set schPath, docPath  
var schemaStr = String(fs.readFileSync(schPath));  
var docStr = String(fs.readFileSync(docPath));  
  
var schema = xsd.parse(schemaStr);  
var validationErrors = schema.validate(docStr);
```

см. <https://www.npmjs.com/package/libxml-xsd>

## Пример валидации файла в Node.js: xsd-schema-validator

```
npm install --save xsd-schema-validator
```

```
var validator = require('xsd-schema-validator'),  
    fs = require('fs');  
  
// set schPath, docPath  
var docStr = String(fs.readFileSync(docPath));  
  
validator.validateXML(docStr, schPath, function(err, res) {  
    if (err) {  
        throw err;  
    }  
  
    res.valid; // true  
});
```

см. <https://www.npmjs.com/package/xsd-schema-validator>

# Парсинг XML, DOM

# SAX парсер

## SAX - Simple API for XML

- Проходит по документу и вызывает коллбеки
  - `startDocument / endDocument`
  - `startElement / endElement`
  - `attribute`
  - `textNode`
- Константен по памяти по количеству тегов
- Полезен, когда есть ограничения по памяти или нужен простой парсер

# Пример парсинга файла в Node.js: sax-js

```
npm install https://github.com/isaacs/sax-js
```

```
var sax = require("sax"),
    fs = require('fs'),
    strict = true, // set to false for html-mode
    parser = sax.parser(strict);

parser.onerror = function (e) {
  // an error happened.
};

parser.ontext = function (t) {
  // got some text. t is the string of text.
};

parser.onopentag = function (node) {
  // opened a tag. node has "name" and "attributes"
};

parser.onattribute = function (attr) {
  // an attribute. attr has "name" and "value"
};

parser.onend = function () {
  // parser stream is done, and ready to have more stuff written to it.
};

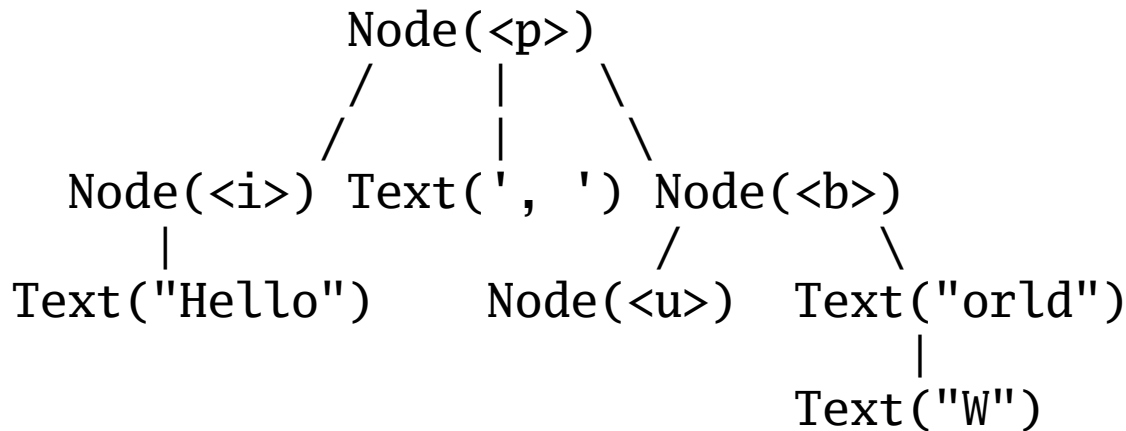
// set xmlPath
var xmlString = String(fs.readFileSync(xmlPath));
parser.write(xmlString).close();
console.log('done'); // works because this is synchronous mode
```

см. <https://github.com/isaacs/sax-js>,  
<https://www.npmjs.com/package/saxes>,  
<https://www.npmjs.com/package/sax-parser>

## DOM - Document Object Model

- Иерархическая объектная модель XML документа
- Занимает место в памяти
- Активно используется в Web

`<p><i>Hello</i>, <b><u>W</u>orld</b></p>`





# Пример парсинга в DOM в браузерах

Класс DOMParser:

```
let parser = new DOMParser();  
let dom = parser.parseFromString('<p><i>Hello</i>,\n<b><u>W</u>orld</b></p>', "text/xml");  
  
dom.children[0] // <p>  
dom.children[0].children[0] // <i>  
dom.children[0].children[0].textContent // "Hello"
```

см. <https://developer.mozilla.org/en-US/docs/Web/API/DOMParser>

# Пример парсинга в DOM в Node.js

```
npm install dom-parser
```

```
let DomParser = require('dom-parser');  
let parser = new DomParser();  
// let xmlString = String(fs.readFileSync(xmlPath));  
let dom = parser.parseFromString('<p><i>Hello</i>,\n<b><u>W</u>orld</b></p>', "text/xml");
```

```
dom.children[0] // <p>  
dom.children[0].children[0] // <i>  
dom.children[0].children[0].textContent // "Hello"
```

см. <https://www.npmjs.com/package/dom-parser>

# Browser DOM API

Работа с деревом:

- `.children` – коллекция детей-элементов
- `.childNodes` – коллекция детей (+ текстовые узлы)
- `.parentNode` – родительский элемент
- `.appendChild`, `.insertBefore`, `.removeChild` – модификация
- `.getAttribute`, `.setAttribute` – работа с атрибутами тэга
- `.textContent`, `.innerHTML`, `.outerHTML` – работа с содержимым
- `.getElementsByTagName` – выбор набора узлов с таким тегом из дерева потомков

Внимание, `NodeList` – массивоподобные живые объекты:

```
var b = document.body, ch = b.children;
ch.forEach(console.log); //Exception
[].forEach.call(ch, console.log.bind(console)); //OK

for(var i=0; i<ch.length; ++i) b.removeChild(ch[i]) //Error
while(ch.length) b.removeChild(ch[0]); //better
[].slice.call(ch).forEach(x => b.removeChild(x)); //OK
```

# Поиск в XML-документе (XPath)

tag2	выбор тега tag2
tag/another tag	выбор another tag, прямой потомок tag
/mydoc/tag	выбор от корня документа
@a	выбор атрибута a
tag//another tag	выбор another tag, любой потомок tag
tag2[last()-1]	выбор предпоследнего tag2
*	выбор любого элемента
@*	выбор любого атрибута
//*	выбор всех элементов
tag   //another tag	выбор и тех, и других

## Пример использования XPath в браузере

```
let parser = ...
let dom = parser.parseFromString(
  '<p><i>Hello</i>', ' +
  '<b><u>W</u>orld</b></p>',
  "text/xml");

//                                query  node                ... type
let ss = dom.evaluate('//*', dom, null,
  XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);

for (let i=0; i<ss.snapshotLength; i++) //snapshotLength=4
  ss.snapshotItem(i); // <p>, <i>, <b>, <u>
```

# Преобразование XML

# XSL Transformations

XSL - eXtensible Stylesheet Language

XSLT - XSL Transformations

- XSL записывается в терминах XML
- XSL - функциональный язык
- XSLT позволяет наложить на XML стили XSL и получить на выходе готовый документ

# Пример XSL преобразования

```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>

</xsl:stylesheet>
```



# XSL сущности

- `<template>` - шаблон, по которому генерируется нечто, для того, что попало под `match` (`match` - XPath-выражение)
- `<value-of>` - вставка выбранного (`select` - XPath-выражение) в вывод
- `<for-each>` - обход набора узлов (`select` - XPath-выражение) и применение того, что в теле
- `<if>`, `<choose>``<when>` - применение того, что в теле, если условие в `test` истинно
- `<apply-templates>` - применение шаблонов (`select` - XPath-выражение, для которого применять шаблоны)
  - `<xsl:template match="/">` - аналог `main` в XSLT
  - `<xsl:apply-templates select="xxx">` - аналог `xxx()` в XSLT

## Пример преобразования в браузере (экспериментальные возможности)

```
// set Node xsl, Node xml

if (!document.implementation
    || !document.implementation.createDocument)
    throw ('Sorry, no XSLT here');

let xsltProc = new XSLTProcessor();

xsltProc.importStylesheet(xsl);

let res = xsltProc.transformToFragment(xml, document);

document.getElementById("example").appendChild(res);
```

# Пример преобразования файлов в Node.js: xslt-processor

```
npm install xslt-processor
```

```
import { xsltProcess, xmlParse } from 'xslt-processor'  
import { readFileSync, writeFileSync } from 'fs'
```

```
// set xmlPath, xsltPath, outputPath
```

```
const xmlString = String(readFileSync(xmlPath));
```

```
const xsltString = String(readFileSync(xsltPath));
```

```
// xmlString: string of xml file contents
```

```
// xsltString: string of xslt file contents
```

```
// outXmlString: output xml string.
```

```
const outXmlString = xsltProcess(  
    xmlParse(xmlString),  
    xmlParse(xsltString)
```

```
);
```

```
writeFileSync(outputPath, outXmlString);
```

см. <https://www.npmjs.com/package/xslt-processor>

# Ссылки

# w3schools tutorial references

- XML, обзор технологий: <https://www.w3schools.com/xml/>
- AJAX: [https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp)
- DOM: [https://www.w3schools.com/xml/dom\\_intro.asp](https://www.w3schools.com/xml/dom_intro.asp)
- XPath: [https://www.w3schools.com/xml/xpath\\_intro.asp](https://www.w3schools.com/xml/xpath_intro.asp)
- XSLT: [https://www.w3schools.com/xml/xsl\\_intro.asp](https://www.w3schools.com/xml/xsl_intro.asp)
- XQuery: [https://www.w3schools.com/xml/xquery\\_intro.asp](https://www.w3schools.com/xml/xquery_intro.asp)
- DTD: [https://www.w3schools.com/xml/xml\\_dtd\\_intro.asp](https://www.w3schools.com/xml/xml_dtd_intro.asp)
- XSD: [https://www.w3schools.com/xml/schema\\_intro.asp](https://www.w3schools.com/xml/schema_intro.asp)

Warning: на w3schools.com тьюториалы, а не спецификации.