

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
  <name>
    <first>Mats</first>
    <last>Roshauw</last>
  </name>
  <email type="private">karmats@gmail.com</email>
  <email type="school">maka0605@student.miun.se</email>
</student>
```

Uppgift 8

PROCESSA XML I JAVASCRIPT

Mats Roshauw | maka0605@student.miun.se | DV019G DATAVETENSKAP GR (B), XML |
2022-06-02

Sammanfattning

Eftersom alla webbsidor bygger på det XML-liknande märkspråket HTML och det faktum att JavaScript kan tolkas i alla webbläsare så har JavaScript bra stöd för att processa XML. Hela DOM API:et finns att tillgå i det globala objektet 'document'. JavaScript har även stöd för XPath och "CSS query selectors", som gör det lätt att hitta det man söker. Att konvertera en XML-textsträng till DOM och vice versa är även det inbyggt. Lite sämre är det med stöd för att validera ett XML-dokument mot en DTD eller XSD. Då måste ett tredjepartsbibliotek användas.

Innehåll

Inledning	3
Exempel	3
DOM	4
Söka i XML	4
DOM API	4
XPath.....	5
Query selector.....	6
Manipulera dokument	6
Skapa element.....	7
Lägga till ett element.....	7
Ersätta ett element med ett annat	8
Ta bort ett element.....	8
Konvertera textsträng till DOM	8
Konvertera DOM till textsträng	9
Validera XML	9
Referenslista	10

Inledning

JavaScript är ett språk som används främst i webbläsare, men kan även användas för server-programmering (Wikipedia contributors 2022). Eftersom webbsidor är uppbyggda av det XML-liknande märkspråket HTML så har JavaScript bra stöd för att läsa och processa XML. Det som tas upp i denna uppsats gäller endast för JavaScript i webbläsare, även om en del av det också stöds för JavaScript på server.

Exempel

Vid kod-exempel av de olika teknikernas kommer följande XML-dokument att användas.

```
<?xml version="1.0" encoding="UTF-8"?>
<starwarsmovies>
  <movie id="1" episode="4">
    <name>A new hope</name>
    <director>George Lucas</director>
    <releasedate>1977-05-25</releasedate>
  </movie>
  <movie id="2" episode="5">
    <name>The empire strikes back</name>
    <director>Irvin Kershner</director>
    <releasedate>1980-05-17</releasedate>
  </movie>
  <movie id="6" episode="3">
    <name>Revenge of the Sith</name>
    <director>George Lucas</director>
    <releasedate>2005-05-19</releasedate>
  </movie>
</starwarsmovies>
```

DOM

Document Object Model (DOM) definierar en standard för tillgång och manipulering av dokument:

"W3C Document Object Model (DOM) är ett plattformsoch språkneutralt gränssnitt som tillåter program och skript att dynamiskt komma åt och uppdatera innehållet, strukturen och stilen i ett dokument." (W3Schools u.å.).

DOM representerar ett dokument som en trädstruktur. Varje gren av trädet slutar i en nod, och varje nod representerar en del av dokumentet (till exempel ett element, textsträng eller kommentar). DOM-metoder tillåter programmatisk åtkomst till trädet. Med dem kan du ändra dokumentets struktur, stil eller innehåll. (MDN contributors 2021).

Söka i XML

Eftersom det i webbutveckling är viktigt att snabbt kunna söka fram olika delar i ett dokument så finns det flera olika sätt att göra det på.

DOM API

Den standardiserade sättet att hämta element på. DOM API stöds i de flesta språk vilket gör att många utvecklare förmodligen redan kan det. Att hitta element via DOM API är dock ganska begränsat vilket gör att man måste ha bra kännedom om hur dokumentet är uppbyggt. DOM API:et är inbyggt i det globala document-objektet.

Exempel:

Hämta elementet med id "1".

```
document.getElementById('1')
```

Hämta alla utgivningsdatum-element

```
document.getElementsByTagName('releasedate')
```

XPATH

XPath står för XML Path Language och används främst för XSLT ([MDN contributors 2022](#)). XPath är väldigt kraftfullt, men kan lätt leda till komplicerade förfrågningar som är svåra att förstå (Testim 2020).

För att söka med XPath i JavaScript så använder man sig av funktionen 'evaluate' som ligger på document-objektet. Funktionen tar fem parametrar (MDN contributors 2022)

- `xpathExpression`: En sträng som innehåller det XPath-uttryck som ska utvärderas.
- `contextNode`: En nod i dokumentet mot vilken `xpathExpression` ska utvärderas, inklusive alla dess undernoder. Dokumentnoden (`document`) är den vanligaste.
- `namespaceResolver`: En funktion som skickas över alla namnrymdsprefix som finns i `xpathExpression` och returnerar en sträng som representerar namnrymdens URI associerad med det prefixet. Detta möjliggör konvertering mellan prefixen som används i XPath-uttrycken och de möjliga olika prefix som används i dokumentet. Om null definieras används inte namnrymder.
- `resultType`: En konstant som anger önskad resultattyp som ska returneras som ett resultat av utvärderingen. Den vanligaste konstanten är `XPathResult.ANY_TYPE` som returnerar resultaten av XPath-uttrycket som den mest naturliga typen.
- `resultat`: Om ett befintligt `XPathResult`-objekt anges, kommer det att återanvändas för att returnera resultaten. Om null anges så skapas ett nytt `XPathResult`-objekt. Om null är definierat så skapas ett nytt resultat-objekt.

Exempel:

Räkna alla filmer

```
document.evaluate('count(//movie)', document, null, XPathResult.ANY_TYPE, null );
```

Hämta alla filmer där regissören heter "George"

```
document.evaluate('//movie[contains(director,"George")]', document, null, XPathResult.ANY_TYPE, null)
```

QUERY SELECTOR

Det vanligaste sättet att hämta element i ett HTML-dokument, kanske inte lika vanligt för XML. Man söker med hjälp av en textsträng med en eller flera CSS-selektorer (MDN contributors 3 2022). Att söka med CSS-selektorer är snabbare än XPath och har en lätt syntax om man kan CSS. Man har även en stor chans att hitta elementet man söker (Testim 2020).

Att söka med CSS-selektorer gör man med hjälp av funktionerna 'querySelector' och 'querySelectorAll' som ligger på det globala document-objektet. Båda funktioner tar en parameter, 'selectors', där man definierar en textsträng som innehåller den eller de CSS-selektorer man vill söka efter. Skillnaden mellan de båda funktionerna är att den förstnämnda returnerar första elementet som matchar, medan den andra returnerar en lista med alla element som matchar (MDN contributors 3 2022).

Exempel:

För att hämta alla namn-element på alla filmer. I exemplet måste elementet 'name' vara ett direkt barn till 'movie'.

```
document.querySelectorAll('movie > name')
```

För att hämta regissörs-elementet för den sjätte episoden

```
document.querySelector('movie[episode="6"] director')
```

Manipulera dokument

Manipulation av ett XML-dokument i JavaScript görs med hjälp av DOM API (MDN contributors 4 2022). Man kan även ersätta noder med att sätta en XML-sträng direkt under en nod via objektsattributet 'innerHTML'.

SKAPA ELEMENT

För att skapa ett element används DOM-funktionen 'createElement'. För att till exempel skapa en ny film med attributen id="3" och episode="6" i exempel XMLen så skriver man

```
const movieElm = document.createElement('movie')
movieElm.setAttribute('id', '3')
movieElm.setAttribute('episode', '6')
```

LÄGGA TILL ETT ELEMENT

Eftersom movie-elementen har tre barn-element så måste även de skapas för att sedan adderas till movie-elementet. Element adderas till andra element via metoderna 'appendChild' eller 'insertBefore'. Nedan visas hur namn-elementet skapas och adderas med hjälp av 'appendChild'.

```
const nameElm = document.createElement('name')
// Skapar en text nod med innehållet för 'nameElm'
const nameTextNode = document.createTextNode('Return of the Jedi')
nameElm.appendChild(nameTextNode)
movieElm.appendChild(nameElm)
```

Ett annat alternativ är att skapa hela innehållet med hjälp av funktionen 'innerHTML'. Risken med att använda sig av 'innerHTML' (Marian Čaikovski 2021) är att den kan innehålla exekverbar kod, medan 'appendChild' måste vara av nod-typ vilket förhindrar att detta sker. Exekverbar kod kan definieras i en CDATA-sektion (Sujatha Paramasivam 2020).

```
movieElm.innerHTML = `<name>Return of the Jedi</name>
<director>Richard Marquand</director>
<releasedate>1983-05-25</releasedate>`
```


ERSÄTTA ETT ELEMENT MED ETT ANNAT

För att ersätta ett element med ett annat används funktionen 'replaceChild'. Funktionen tar två parametrar, varav den första är en existerande barn-nod och den andra ett skapat element. I nedan exempel ersätts filmen "Revenge of the Sith" med "Return of the Jedi".

```
const rotsElm = document.getElementById('6')
document.getElementsByTagName('starwarsmovies')[0].replaceChild(movieElm, rotsElm)
```

TA BORT ETT ELEMENT

Ta bort element görs med funktionen 'removeChild'. Funktionen tar en parameter, det element som ska tas bort. I exemplet nedan tas filmen "A new hope" bort från dokumentet.

```
const anhElm = document.getElementById('1')
document.getElementsByTagName('starwarsmovies')[0].removeChild(anhElm)
```

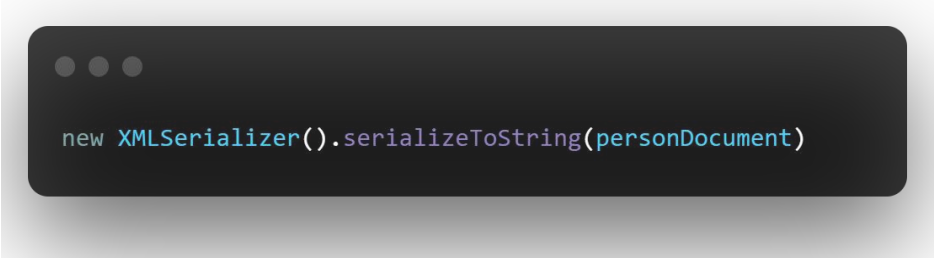
Konvertera textsträng till DOM

För att konvertera en textsträng till DOM används DOMParser. DOMParser har endast en metod 'parseFromString' som tar två argument, texten som ska analyseras och vilken MIME-typ strängen antas ha (MDN contributors 5 2022). För att analysera XML används MIME-typen "text/xml". I nedanexempel skapas ett DOM-dokument, 'person', med ett barn, 'name', med en text-nod med värdet "Mats Roshauw".

```
const personXmlString = `<person>
  <name>Mats Roshauw</name>
</person>`
const personDocument = new DOMParser().parseFromString(personXmlString, 'text/xml')
```

Konvertera DOM till textsträng

För att konvertera DOM till textsträng så används XMLSerializer. XMLSerializer har även den endast en metod 'serializeToString' som inte tar några argument. Nedan exempel konverterar det skapade person-dokumentet till en textsträng.

A dark-themed code editor window with three small circles in the top-left corner. It contains a single line of JavaScript code: `new XMLSerializer().serializeToString(personDocument)`.

```
new XMLSerializer().serializeToString(personDocument)
```

Validera XML

Det stöds inte att validera XML mot ett schema i JavaScript, men det finns flera olika tredjeparts-bibliotek som tillhandahåller sådan funktionalitet.

Referenslista

Wikipedia contributors (2022) *List of server-side JavaScript implementations*
https://en.wikipedia.org/wiki/List_of_server-side_JavaScript_implementations [2022-05-30]

W3Schools (u.å.) *XML DOM* https://www.w3schools.com/xml/xml_dom.asp [2022-05-31]

MDN contributors (2021) *DOM (Document Object Model)*
<https://developer.mozilla.org/en-US/docs/Glossary/DOM> [2022-06-01]

MDN contributors (2022) *XPath* <https://developer.mozilla.org/en-US/docs/Web/XPath> [2022-06-01]

Testim (2020) *XPath vs CSS Selector: The Difference and How to Choose*
<https://www.testim.io/blog/xpath-vs-css-selector-difference-choose> [2022-06-01]

MDN contributors 2 (2022) *Introduction to using XPath in JavaScript*
https://developer.mozilla.org/en-US/docs/Web/XPath/Introduction_to_using_XPath_in_JavaScript [2022-06-01]

MDN contributors 3 (2022) *Document.querySelector()* <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector> [2022-06-01]

MDN contributors 4 (2022) *Node* <https://developer.mozilla.org/en-US/docs/Web/API/Node> [2022-06-02]

Marian Čaikovski (2021) *innerHTML vs appendChild()* <https://marian-caikovski.medium.com/innerHTML-vs-appendchild-e74c763846df> [2022-06-02]

Sujatha Paramasivam (2020) *XML cross-site scripting check* <https://docs.citrix.com/en-us/citrix-adc/current-release/application-firewall/xml-protections/xml-cross-site-scripting-check.html> [2022-06-02]

MDN contributors 5 (2022) *DOMParser.parseFromString()*
<https://developer.mozilla.org/en-US/docs/Web/API/DOMParser/parseFromString> [2022-06-02]