XML e AJAX

Introduzione
ai
concetti base

Dott. Ing. Ivan Ferrazzi
V1.2 del 15/01/2014

Indice generale

XML

Introduzione

XML (eXtensible Markup Language) è un linguaggio che si basa sull'utilizzo di tag (es. `<tag></tag>`) simili a quelli utilizzati in HTML. L'idea di fondo è stata quella di creare un sistema con il quale fosse possibile trasportare dei dati indipendentemente dal programma utilizzato. XML utilizza infatti i tag per identificare il contenuto ed il tipo di dati da trasportare piuttosto che occuparsi di una loro forma di visualizzazione.
La flessibilità di XML sta nel fatto di non avere dei tag predefiniti come ad esempio nel linguaggio HTML, ma possono essere creati in base alle proprie esigenze. In poche parole XML non fa nulla (da un punto di vista di esecuzione), ma dà la possibilità ai programmi di interpretarne in maniera corretta il suo contenuto.

Perché utilizzare XML

Nello sviluppo di pagine web possiamo sfruttare XML per dividere i dati effettivi dal linguaggio HTML. In questa maniera possiamo utilizzare HTML per concentrarci esclusivamente sul dove e come vengono visualizzati i dati. Con poche righe di codice JavaScript possiamo poi importare i dati necessari all'interno dei blocchi presenti nel codice HTML.
XML non si limita solamente all'utilizzo nell'ambito del web, ma può essere utilizzato da programmi applicativi come strumento per esportare od importare dati tra versioni diverse dello stesso programma o addirittura da programmi completamente diversi. L'esportazione e

l'importazione dei dati da un programma ad un altro può rivelarsi un lavoro molto dispendioso da un punto di vista di tempo. Le problematiche dipendono spesso e volentieri dai vari formati utilizzati per la memorizzazione dei dati. XML risolve questo problema utilizzando un sistema indipendente da una qualsiasi forma di hardware o software.
Ecco perché oggigiorno viene utilizzato molto per trasportare dei dati attraverso l'internet oppure tra sistemi operativi diversi. Addirittura ci sono dei formati utilizzati per il salvataggio di documenti che si basano su XML (es. Open Document Type).
Con XML possiamo addirittura creare dei nuovi linguaggi per l'internet come ad esempio:

- XHTML
- WSDL per la descrizione di nuovi servizi web disponibili
- WAP e WML per apparecchiature mobile
- linguaggi RSS per le news

La struttura dei documenti XML

Vediamo ora un esempio di documento XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rubrica>
  <contatto>
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <telefono>123456</telefono>
  </contatto>
  <contatto>
    <cognome>Verdi</cognome>
    <nome>Antonella</nome>
    <telefono>223344</telefono>
  </contatto>
</rubrica>
```

La prima riga identifica il tipo di documento XML che stiamo utilizzando. L'attributo `version` identifica la versione mentre `encoding` identifica il tipo di charset utilizzato per la memorizzazione del testo (`ISO-8859-1` equivale al charset `Latin-1/West European`).
Ogni elemento all'interno del documento è composto da un tag iniziale `<...>`, che ne identifica l'inizio, e un tag finale `</...>`, che ne identifica la fine.
L'elemento `rubrica` definisce l'elemento root del documento (**root element**) che contiene 2 elementi figlio (**child elements**) identificati come `contatto`. Gli elementi `contatto` contengono a loro volta altri 3 elementi figlio, ossia `cognome`, `nome` e `telefono` per ogni elemento contatto.

Un elemento può contenere, oltre al nome che lo identifica, un valore oppure degli attributi che ne descrivono le proprietà. Il valore, come

abbiamo già visto nell'esempio precedente, viene scritto tra il tag iniziale e finale dell'elemento. L'attributo, invece, viene integrato all'interno del tag stesso e ha la forma

```
nome="valore"
```

oppure

```
nome='valore'
```

Il valore viene scritto sempre tra virgolette (") oppure apici ('). Se volessimo, ad esempio, definire la lingua per ogni contatto potremmo modificare l'esempio precedente come segue:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rubrica>
  <contatto lingua="it">
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <telefono>123456</telefono>
  </contatto>
  <contatto lingua="de">
    <cognome>Verdi</cognome>
    <nome>Antonella</nome>
    <telefono>223344</telefono>
  </contatto>
</rubrica>
```

In questo caso `lingua` diventa l'attributo dell'elemento contatto. Nel linguaggio XML non viene specificato quando è meglio definire un valore come attributo di un elemento o come nuovo sotto elemento. Per definire la lingua del contatto potremmo scrivere anche come segue

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rubrica>
  <contatto>
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <telefono>123456</telefono>
    <lingua>it</lingua>
  </contatto>
  <contatto>
    <cognome>Verdi</cognome>
    <nome>Antonella</nome>
    <telefono>223344</telefono>
    <lingua>de</lingua>
  </contatto>
</rubrica>
```

Un altro vantaggio molto importante di XML è la sua flessibilità nell'aggiunta di elementi all'interno di documenti già in uso. Programmi in grado di leggere documenti XML ne leggono l'intero contenuto, ma utilizzano solamente gli elementi che conoscono. In questo caso

possiamo quindi aumentare il documento a piacere senza causare errori all'interno del programma in esecuzione.

La sintassi

Come abbiamo già visto XML obbliga l'utilizzo dei tag finali per ogni elemento utilizzato. La definizione del seguente elemento non sarebbe quindi corretta

```
<cognome>Rossi
```

ma bisogna utilizzare

```
<cognome>Rossi</cognome>
```

Inoltre bisogna stare attenti alle proprietà *case sensitive* del linguaggio stesso. Il nome utilizzato come tag iniziale deve corrispondere esattamente al nome utilizzato come tag finale. Non possiamo quindi scrivere

```
<Cognome>Rossi</cognome>
```

ma bisogna utilizzare

```
<cognome>Rossi</cognome>
```

Quando scegliamo un nome che dovrà identificare un nostro elemento dobbiamo stare attenti ad alcune regole definite all'interno del linguaggio XML. I nome degli elementi:

- possono contenere lettere, numeri ed altri caratteri;
- non possono iniziare con numeri o caratteri di punteggiatura;
- non possono iniziare con una qualsiasi forma minuscola o maiuscola di xml, quindi xml, Xml, XML, ecc.
- non possono contenere degli spazi

Un'altra questione molto importante è l'ordine di apertura e di chiusura di elementi e sotto elementi. Non possiamo chiudere un elemento se non sono prima stati chiusi tutti i sui sotto elementi. Vediamo un esempio

```
<b><i>il mio testo</b></i>
```

Questa forma non è corretta perché viene chiuso il tab `<b>` mentre `<i>` (sotto elemento di `<b>`) rimane ancora aperto. Per questo motivo dobbiamo scrivere

```
<b><i>il mio testo</i></b>
```

Uno degli elementi più importanti all'interno di un documento XML è **l'elemento root**. Questo è l'unico elemento che non può ripetersi all'interno del documento. L'elemento root è infatti l'elemento principale che identifica il contenuto dell'intero documento XML. Gli elementi che si trovano all'interno dell'elemento root possono ripetersi per descrivere situazioni diverse, esattamente come lo abbiamo visto all'interno del nostro esempio. La nostra rubrica contiene due contatti.

In alcuni casi diventa indispensabile utilizzare come valore degli elementi dei testi che XML utilizza per interpretare la struttura del documento stesso. Proviamo a pensare al carattere < oppure al carattere >. Il seguente elemento genererebbe un errore XML

```
<controllo>anni < 18</controllo>
```

Per evitare questi problemi possiamo utilizzare dei codici che vengono interpretati da XML come caratteri speciali. I 5 codici predefiniti per rappresentare caratteri speciali sono

`<`	<	minore di
`>`	>	maggiore di
`&`	&	e commerciale
`&apos;`	'	apice
`&quot;`	"	doppie virgolette

Per quanto riguarda gli spazi utilizzati all'interno dei valori degli elementi non dobbiamo confonderci con quello che è il comportamento di HTML. In XML ogni spazio identifica esattamente uno spazio, mentre in HTML vediamo trasformarsi una serie di spazi in un unico spazio visualizzato all'interno del nostro browser.

XML ci mette a disposizione la possibilità di inserire dei commenti all'interno del nostro documento per migliorane la leggibilità. Un commento viene inserito come segue

```
<!-- questo è il nostro commento -->
```

XML Validation

Un programma in grado di leggere un documento XML deve essere in grado di verificare la validità del documento stesso. La struttura adottata deve rispettare delle regole che definiamo e solo in questo caso possiamo parlare di un documento valido. Il rendere un documento valido viene comunemente chiamato XML Validation.
Le regole che intendiamo adottare vengono definite all'interno di un DTD, ossia Document Type Definition. Possiamo quindi modificare il nostro esempio come segue

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rubrica SYSTEM "rubrica.dtd">
<rubrica>
  <contatto lingua="it">
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <telefono>123456</telefono>
  </contatto>
  <contatto lingua="it">
    <cognome>Verdi</cognome>
    <nome>Antonella</nome>
    <telefono>223344</telefono>
  </contatto>
</rubrica>
```

Con la riga aggiunta definiamo il tipo di documento `rubrica` che si basa sulle regole inserite all'interno del file esterno `rubrica.dtd`. Vediamo ora il contenuto del file in questione:

```
<!ELEMENT rubrica (contatto+)>
<!ELEMENT contatto (cognome,nome,telefono)>
<!ATTLIST contatto lingua CDATA>
<!ELEMENT cognome (#PCDATA)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT telefono (#PCDATA)>
```

In questo caso possiamo definire con `<!ELEMENT …>` gli elementi che utilizziamo all'interno del file XML, mentre possiamo definire gli attributi presenti all'interno dell'elemento con `<!ATTLIST ...>`. L'elemento `contatto` è composto dagli elementi definiti all'interno delle parentesi tonde. Per quanto riguardi i singoli elementi questi devono essere definiti a loro volta all'interno di un `<!ELEMENT ...>`.

`<!ELEMENT ...>`
Questo blocco identifica il nome dell'elemento da descrivere e tra parentesi tonde ne definisce la struttura. La struttura può essere definita come `#PCDATA` (combinazione mista di elementi, oppure un valore di testo), un elenco di sotto elementi presenti all'interno di esso, oppure una combinazione dei due tipi.
La presenza di un `+`, `*`, oppure `?`, dietro al nome nell'elenco dei sotto elementi definisce rispettivamente se l'elemento deve essere presente una o più volte, mai o più volte, oppure mai o una volta.

`<!ATTLIST ...>`
Questo blocco permette di identificare le proprietà di un attributo all'interno di un elemento. La forma che utilizziamo è la seguente

`<!ATTLIST` *`elemento attributo tipo_attributo {"default"} {stato}`*`>`

`elemento`	Il nome dell'elemento per il quale si sta definendo l'attributo

`attributo`	Il nome dell'attributo che si sta definendo
`tipo_attributo`	Come tipo di attributo possiamo trovare `CDATA` (che definisce come valore un testo), `ID` (che identifica un valore univoco all'interno del documento XML), `IDREF` (l'id utilizzato fa riferimento all'id di un altro elemento), `IDREFS` (che identifica un elenco di id utilizzate), `NMTOKEN` (identifica il nome di un elemento valido in XML), `NMTOKENS` (elenco di nomi XML), `ENTITY` (identifica un'entità definita), `ENTITIES` (un elenco di entità), `NOTATION` (il nome di una notazione), `xml:` (un valore predefinito di XML), oppure una serie di possibili valori all'interno di parentesi tonde separate dal simbolo pipe (|), es `(si | no)`.
`default`	Il valore di default nel caso in cui non venisse definito.
`stato`	I valori possibili possono essere `#REQUIRED` (l'attributo deve essere sempre presente all'interno di ogni elemento), `#IMPLIED` (l'attributo è opzionale), oppure `#FIXED` *value* (il valore è fisso e non può essere modificato).

Vediamo ora un semplice esempio:

```
<!ATTLIST contatto lingua (it | de) "it" #IMPLIED>
```

Se apriamo ora il file XML creato con XML-Parser vedremo apparire la struttura del nostro file, ma priva di ogni tipo di formattazione. XML ci dà la possibilità di integrare un file CSS da utilizzare per l'eventuale formattazione all'interno di un parse. In questo caso aggiungiamo la seguente riga:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rubrica SYSTEM "rubrica.dtd">
<?xml-stylesheet type="text/css" href="rubrica.css"?>
<rubrica>
  <contatto>
    <cognome>Rossi</cognome>
    <nome>Mario</nome>
    <telefono>123456</telefono>
  </contatto>
  <contatto>
    <cognome>Verdi</cognome>
    <nome>Antonella</nome>
    <telefono>223344</telefono>
  </contatto>
</rubrica>
```

e creiamo il file `rubrica.css` con il seguente contenuto:

```
rubrica {
  background-color: #eeeeee;
  width: 100%;
```

```
}
contatto {
  display: block;
  margin-bottom: 10pt;
  margin-left: 10pt;
}
cognome,nome {
  color: #0000ff;
  font-size: 18pt;
}
telefono {
  color: #000000;
  font-size: 12pt;
}
```

AJAX

Introduzione

AJAX sta per **Asynchronous Javascript And Xml** e non identifica un nuovo linguaggio di programmazione ma bensì un nuovo modo di utilizzare JavaScript per caricare pagine senza dover ricaricare l'intera pagina HTML esistente.
L'oggetto che mette a disposizione questo tipo di funzionalità è `XMLHttpRequest` (oppure `ActiveXObject` per IE5 e IE6).

Richieste con AJAX

Per poter recuperare le informazioni che ci interessano dobbiamo quindi creare l'oggetto in questione e definire i parametri per l'invio della richiesta in questione.
Vediamo un semplice esempio per creare l'oggetto con le operazioni per la richiesta qui di seguito

```
if(window.XMLHttpRequest) {
  //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}else{
  //crea un parser in IE5, IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("POST","rubrica.php",false);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-
```

```
urlencoded");
xmlhttp.send("id=235");
```

L'oggetto viene registrato all'interno della variabile `xmlhttp` che ci metterà a disposizione le funzioni `open()`, `setRequestHeader()` e `send()`.

`open(metodo,url,async)`
Con questa funzione possiamo definire il metodo da utilizzare per l'invio di della richiesta (`GET` oppure `POST`), l'`url` del file che vogliamo aprire sul server e `async` (`true` per l'invio asincrono, oppure `false` per l'invio sincrono).

`setRequestHeader(header,valore)`
Questa funzione permette di modificare il header che verrà mandato insieme alla richiesta del file definito all'interno della funzione precedente. `header` contiene il nome del header da utilizzare, mentre `valore` ne contiene il valore.

`send(stringa)`
Questa funzione invia quanto definito. La stringa viene utilizzata come elenco delle copie (`nome=valore`) separate dalla e commerciale (`&`) per l'invio di parametri mediante metodo `POST`. Nel caso in cui non si utilizzasse `POST` ma `GET` questa stringa la si dovrebbe definire come stringa vuota (ossia `""`).

Una volta inviata la richiesta dobbiamo avere la possibilità di recuperare la risposta. Abbiamo a disposizione due funzioni che ci permettono di recuperare il risultato come stringa (`responseText`) oppure come struttura XML (`responseXML`).

La risposta come stringa

La risposta che riceviamo la possiamo recuperare come stringa. In questo caso possiamo utilizzare il risultato per andare a riempire un elemento presente all'interno della pagina come segue:

```
if(window.XMLHttpRequest) {
  //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}else{
  //crea un parser in IE5, IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","rubrica.php",false);
xmlhttp.send("");
testo=xmlhttp.responseText;

document.getElementById("nome").innerHTML=testo;
```

La risposta come parser XML

Oggigiorno ogni moderno browser ha al suo interno un parser XML. Un parser XML è in grado di trasformare un documento XML in un oggetto XML DOM (Document Object Model) che può poi essere manipolato con JavaScript.
Il seguente codice JavaScript interpreta il file XML generando un oggetto XML DOM:

```
if(window.XMLHttpRequest) {
  //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}else{
  //crea un parser in IE5, IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","rubrica.xml",false);
xmlhttp.send("");
xmlDoc=xmlhttp.responseXML;
```

Con window.XMLHttpRequest all'interno del blocco if controlliamo l'esistenza di questo oggetto (non presente per i browser IE5 e IE6). Per creare un nuovo oggetto di parser XML scriviamo quindi

```
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

per i browser IE5 e IE6, che permette di definire l'oggetto ActiveX (`Microsoft.XMLHTTP`) da utilizzare, mentre per tutti gli altri utilizziamo la forma

```
xmlhttp=new XMLHttpRequest();
```

Una volta creato l'oggetto definiamo come e cosa vogliamo aprire con

```
xmlhttp.open("GET","rubrica.xml",false);
```

per poi mandarla via senza utilizzare parametri GET (inviamo come parametro una stringa vuota) con

```
xmlhttp.send("");
```

L'intero risultato di questa richiesta sotto forma di oggetto XML DOM viene recuperato e poi copiato come oggetto all'interno della variabile `xmlDoc` come segue

```
xmlDoc=xmlhttp.responseXML;
```

Un oggetto XML DOM può essere recuperato da un file come nell'esempio appena citato, ma è possibile utilizzare anche semplicemente una stringa. Vediamo il prossimo esempio

```
txt = "<rubrica>";
txt = txt + "<contatto>";
txt = txt + "<cognome>Rossi</cognome>";
txt = txt + "<nome>Mario</nome>";
txt = txt + "<telefono>123456</telefono>";
txt = txt + "</contatto>";
txt = txt + "</rubrica>";

if(window.DOMParser) {
  parser=new DOMParser();
  xmlDoc=parser.parseFromString(txt,"text/xml");
}else{
  //Internet Explorer
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async="false";
  xmlDoc.loadXML(txt);
}
```

In questo esempio recuperiamo l'oggetto XML DOM dal contenuto della stringa `txt`. Anche qui dobbiamo differenziare tra browser Internet Explorer e resto del mondo, dove per Internet Explorer si crea un oggetto ActiveX

```
xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
```

che mette a disposizione la funzione `loadXML(stringa)` in grado di restituire l'oggetto XML DOM in base alla stringa fornita. Negli altri browser si crea un oggetto DOMParser

```
parser=new DOMParser();
```

che mette a disposizione la funzione `parseFromString(stringa,tipo)` in grado di creare l'oggetto XML DOM in base alla stringa ed il tipo di contenuto della stringa (`text/xml`) fornito.

L'oggetto XML DOM

Una volta creato l'oggetto XML DOM dobbiamo riuscire ad estrarne il contenuto. L'oggetto XML DOM rappresenta un documento XML in una struttura ad albero. Ogni elementi all'interno di questo oggetto può essere modificato od eliminato, ma è possibile aggiungere anche nuovi elementi. Tutti gli elementi, gli attributi e i valori degli elementi all'interno di questo oggetto vengono chiamati **nodi**.

Vediamo ora un completo esempio che ci permette di estrarre i dati presenti all'interno del nostro file `rubrica.xml`

```
<html>
  <head>
  </head>
```

```
  <body>
    <div>
      <b><span id="cognome"></span> <span id="nome"></span></b><br />
      <span id="telefono"></span><br />
    </div>

    <script type="text/javascript">
      if(window.XMLHttpRequest) {
        //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
      }else{
        //crea un parser in IE5, IE6
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.open("GET","rubrica.xml",false);
      xmlhttp.send("");
      xmlDoc=xmlhttp.responseXML;

      document.getElementById("cognome").innerHTML=
       xmlDoc.getElementsByTagName("cognome")[0].childNodes[0].nodeValue;
      document.getElementById("nome").innerHTML=
       xmlDoc.getElementsByTagName("nome")[0].childNodes[0].nodeValue;
      document.getElementById("telefono").innerHTML=
       xmlDoc.getElementsByTagName("telefono")[0].childNodes[0].nodeValue;
    </script>
  </body>
</html>
```

Tutti gli elementi che hanno lo stesso nome, ma che appartengono a elementi diversi vengono aggiunti all'interno di un primo array. Questo tag potrebbe però essere presente più volte all'interno dello stesso elemento. Per questo motivo ogni posizione all'interno del primo array dovrà contenere a sua volta un nuovo array `childNodes` che, a partire da 0, conterrà i tag uguali presenti all'interno dello stesso elemento.
Usa come indice del primo array (recuperato direttamente da `getElementsByTagName()`) il valore 0 per recuperare il primo elemento presente all'interno del file XML con il nome desiderato. L'indice di valore 1 restituisce il secondo gruppo di tag presente all'interno del file XML.
Per recuperare il cognome del primo contatto usiamo

```
xmlDoc.getElementsByTagName("cognome")[0].childNodes[0].nodeValue;
```

per recuperare il cognome del secondo invece

```
xmlDoc.getElementsByTagName("cognome")[1].childNodes[0].nodeValue;
```

`xmlDoc.getElementsByTagName()` è quindi un oggetto `array` con all'interno i valori dei vari tag estratti dal file stesso. Possiamo recuperare il numero complessivo degli elementi presenti (nel nostro caso il numero dei contatti) con

```
xmlDoc.getElementsByTagName("cognome").length;
```

Gli eventuali attributi all'interno dell'elemento in questione possono essere estratti con

```
attr=xmlDoc.getElementsByTagName("contatto")[0].getAttribute("lingua");
```

Naturalmente abbiamo anche la possibilità di modificare un qualsiasi elemento presente all'interno dell'oggetto creato. Possiamo quindi modificare il valore di un elemento o di un attributo rispettivamente utilizzando le seguenti forme

```
xmlDoc.getElementsByTagName("cognome")[0].childNodes[0].nodeValue="Verdi";
xmlDoc.getElementsByTagName("contatto")[0].getAttribute("lingua")="de";
```

Aggiungere nuovi attributi all'oggetto XML DOM

Con la funzione `setAttribute(nome_attributo,valore_attributo)` del nostro oggetto è possibile aggiungere un nuovo attributo ad un elemento dove `nome_attributo` identifica il nome del nuovo attributo, mentre `valore_attributo` ne identifica il valore. Nel seguente esempio aggiungiamo l'attributo `lingua` a tutti gli elementi `contatto` presenti all'interno del nostro oggetto dandogli come valore `it`

```
elemento=xmlDoc.getElementsByTagName("contatto");
for(i = 0; i < e.length; i++) {
  elemento[i].setAttribute("lingua","it");
}
```

Modificare la struttura dell'oggetto XML DOM

Con la funzione `createElement(nome_elemento)` creiamo il nuovo nodo *elemento* da inserire, con `createTextNode(valore)` creiamo un nuovo nodo di *testo*, mentre con `appendChild(nodo)` aggiungiamo un nuovo nodo *figlio* come ultimo nodo di un nodo esistente. Nel prossimo esempio aggiungiamo un nuovo elemento `cellulare` all'interno del primo contatto

```
nuovo_el=xmlDoc.createElement("cellulare");
nuovo_testo=xmlDoc.createTextNode("333 101010");
nuovo_el.appendChild(nuovo_testo);

xmlDoc.getElementsByTagName("contatto")[0].appendChild(nuovo_el);
```

Rimuovere un elemento dall'oggetto XML DOM

Con la funzione `removeChild(oggetto_child)` è possibile rimuovere un

elemento dalla struttura dell'oggetto in questione

```
e=xmlDoc.getElementsByTagName("contatto")[0];
e.removeChild(e.childNodes[0]);
```

JSON e AJAX

`JSON` è un oggetto nativo in Javascript e permette di elaborare elementi JSON. In questo caso possiamo recuperare una stringa con sintassi JSON mediante AJAX e convertire poi quanto ricevuto in oggetto JSON grazie al metodo `parse()` del medesimo oggetto.
Creiamo il file `data.json` con all'interno la seguente struttura:

```
{
  “cognome”: “Rossi”,
  “nome”: “Mario”
}
```

Poi creiamo il seguente file html che possiamo chiamare `json.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JSON</title>
    <meta charset=”utf-8” />
    <style>
      function caricaJSON() {
        if(window.XMLHttpRequest) {
          //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
          xmlhttp=new XMLHttpRequest();
        }else{
          //crea un parser in IE5, IE6
          xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.open("GET","data.json",false);
        xmlhttp.send("");

        json_obj = JSON.parse(xmlhttp.responseText);
        alert(json_obj.cognome + “ “ + json_obj.nome);
      }
    </style>
  </head>
  <body>
    <button onclick=”caricaJSON()”>carica</button>
  </body>
</html>
```

La riga

```
json_obj = JSON.parse(xmlhttp.responseText);
```

si occupa di creare l'oggetto dalla stringa recuperata, mentre riusciamo ad accedere ai vari contenuti con

`json.cognome`

oppure

`json.nome`

Richiesta asincrona o sincrona

Le richieste possono essere mandate in maniera asincrona (JavaScript continua ad eseguire i comandi presenti dopo la richiesta) oppure sincrona (JavaScript aspetta di ricevere la risposta prima di proseguire con l'elaborazione delle ulteriori operazioni).
In caso di modalità asincrona send() richiama ad ogni operazione in corso la funzione definita all'interno della variabile `onreadystatechange`, passando con `readyState` il codice dell'attuale operazione in corso e con `status` lo stato dell'attuale richiesta. Nella seguente tabella vediamo i vari valori utilizzati.

`readyState`	Lo stato dell'operazione in corso viene definito come valore numerico, dove 0: la richiesta non è ancora stata inizializzata 1: connessione col server stabilita 2: richiesta ricevuta 3: la richiesta viene elaborata 4: la richiesta è terminata e la risposta è pronta
`status`	200: “OK” 404: Pagina non trovata

Vediamo ora un semplice esempio

```
if(window.XMLHttpRequest) {
  //crea un parser in IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}else{
  //crea un parser in IE5, IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}

xmlhttp.onreadystatechange=
  function() {
    if(xmlhttp.readyState==4 && xmlhttp.status==200) {
      testo=xmlhttp.responseText;
      document.getElementById("nome").innerHTML=testo;
    }
  }

xmlhttp.open("GET","rubrica.php",true);
xmlhttp.send("");
```