La rappresentazione dei dati nel web: XML e JSON

Rev Digitale 2.1 del 18/09/2017

La rappresentazione dei dati nel web

Charset …… 2
Internet Media Types …… 2
Gli oggetti LocalStorage e SessionStorage …… 3
XML …… 3
Navigazione di un albero XML …… 4
Cenni alle regular Expression in Java Script …… 7
Vettori Associativi …… 8
Java Script Object Notation …… 8
Note sugli Oggetti in js …… 11
Utilizzo dei metodi …… 11
Vettori di Object …… 12
Le specifiche di JSON …… 12
Approfondimenti …… 14

Charset

I file html vengono normalmente salvati in formato **UTF-8 senza BOM** (intestazione).
UTF-8 è un formato che salva i caratteri ascii su un byte ed i rimanenti caratteri su 2 bytes e dunque riconosce le lettere accentate. L'intestazione (BOM) serve ad avvisare il lettore riguardo al formato del file però può interferire con la gestione server per cui normalmente non viene usata.
Per cui il formato utilizzato viene scritto di solito in testa al file tramite apposito meta tag:

```
<meta charset="UTF-8">
```

Internet Media Types

Il **media-type** indica il tipo di informazioni contenute all'interno del file :

```
<meta http-equiv="content-type" content="text/html”>
```

IANA manages the official registry of **media types** The identifiers were originally defined in **RFC 2046**, and were called **MIME types** because they referred to the non-ASCII parts of email messages that were composed using the MIME specification (**Multipurpose Internet Mail Extensions**).
They are also sometimes referred to as **Content-types**.
Their use has expanded from **email** sent through SMTP, to other protocols such as HTTP, and others.
New media types can be created with the procedures outlined in **RFC 6838**.

Text Type, for human-readable text and source code.

text/plain: Textual data; Defined in RFC 2046 and RFC 3676
text/html: HTML; Defined in RFC 2854
text/css: Cascading Style Sheets; Defined in RFC 2318
text/xml: Extensible Markup Language; Defined in RFC 3023
text/csv: Comma-separated values; Defined in RFC 4180
text/rtf: RTF; Defined by Paul Lindner
Elenco completo : http://en.wikipedia.org/wiki/Internet_media_type

text/javascript JavaScript; Defined in and made obsolete in RFC 4329 in order to discourage its usage in favor of **application/javascript.** However, text/javascript is allowed in HTML 4 and 5 and, unlike application/javascript, has cross-browser support.
application/json dati JSON serializzati

The "type" attribute of the <script> tag in HTML5 is optional and there is no need to use it at all since all browsers have always assumed the correct **default,** even before HTML5.

Le stringhe su righe multiple

Le stringhe in JS, sia quelle racchiuse tra apici singoli sia quelle racchiuse tra apici doppi, possono essere scritte su righe multiple terminandole con il carattere backslash **** che però deve necessariamente essere l'ultimo carattere della riga, **senza eventuali spazi successivi**.

```
var s = “salve \
        mondo”;
```

Gli oggetti localStorage e SessionStorage

L'oggetto **localStorage** consente di salvare fino a 5 MB di informazioni sul **HD** del PC locale.

L'oggetto **sessionStorage** salva invece i dati soltanto nella memoria del browser, per cui vengono persi nel momento in cui il browser viene chiuso e risultano disponibili soltanto per la pagina che li ha creati.

Entrambi utilizzano la sintassi degli Array Associativi:

```
localStorage.setItem("key1", "value1");
var key1 = localStorage.getItem("key1");
localStorage.removeItem("key1");
localStorage.clear();  // ripulisce l'intera localStorage
localStorage.length;   //  numero di chiavi memorizzate
```

I dati sono sempre salvati come stringhe per cui poi devono essere eventualmente riconvertiti.
Per salvare un intero albero XML o un oggetto JSon occorre prima serializzarli.

Per vedere se il browser supporta la localStorage si può utilizzare una delle seguenti condizioni:

```
if (typeof(localStorage) != "undefined") {
if('localStorage' in window && window['localStorage'] !== null) {
```

XML

Un documento XML è un documento strutturato a tag esattamente come i documenti HTML.
E' sostanzialmente un albero con una radice (equivalente al body html) con dei tag interni di primo livello che possono contenere al loro interno dei valori oppure dei tag di secondo livello.
Ogni tag può avere uno o più attributi.

```
<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>
	<book category="web">
		<title lang="en">XQuery Kick Start</title>
		<author>James McGovern</author>
		<author>Per Bothner</author>
		<author>Kurt Cagle</author>
		<author>James Linn</author>
		<author>Vaidyanathan Nagarajan</author>
		<year>2003</year>
		<price>49.99</price>
	</book>
	<book category="web" cover="paperback">
		<title lang="en">Learning XML</title>
		<author>Erik T. Ray</author>
		<year>2003</year>
		<price>39.95</price>
	</book>
</bookstore>
```

I metodi di lettura da file normalmente leggono il file in modo testuale e ne restituiscono il contenuto all'interno di una stringa :

```
var xml = readFile("bookstore.xml");
alert (xml);
```

Per ragioni di sicurezza i browser impediscono a javascript l'accesso al file system della macchina locale, per cui la riga procedente non può essere eseguita a meno di utilizzare un localStorage.

Il processo di trasformazione di una stringa in un oggetto si definisce **parsing**.
Il processo inverso di trasformazione di un oggetto in stringa si definisce **serializzazione**

Parsing di una stringa XML in un oggetto XML

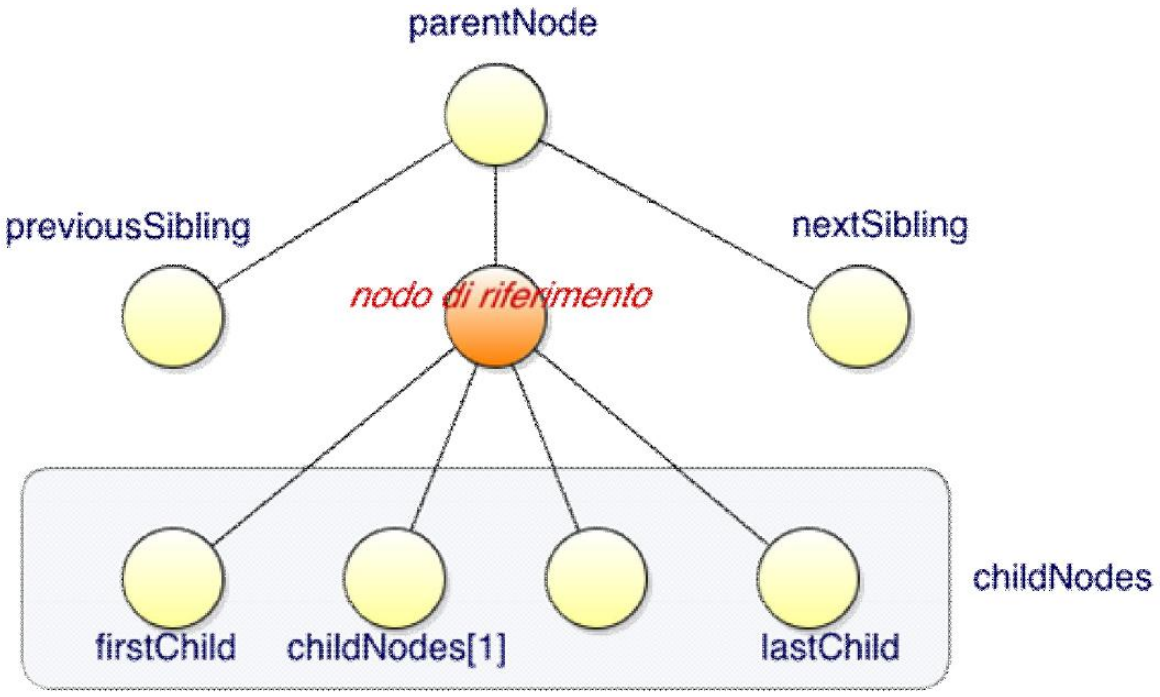
```
var parser=new DOMParser();
var xmlDoc=parser.parseFromString(xml,"text/xml");
```

Serializzazione di un oggetto XML in stringa

```
var serializer = new XMLSerializer();
var xml = serializer.serializeToString(xmlDoc);
```

Navigazione di un albero XML

Un oggetto XML è un albero esattamente come il DOM di una pagina HTML ed è pertanto navigabile tramite gli stessi metodi e le stesse proprietà.



I vari Tipi di nodo

I vari nodi possono essere di diversi tipi. I principali sono i seguenti :

1 = ELEMENT_NODE = nodo vero e proprio
2 = ATTRIBUTE_NODE = attributo
3 = TEXT_NODE = nodo testuale, cioè foglia dell'albero (valore di un nodo ELEMENT)
8 = COMMENT_NODE = commento
9 = DOCUMENT_NODE = l'intero xmlDoc subito dopo il parsing

Proprietà per la navigazione dell'albero (e che restituiscono sempre un Nodo)

.childNodes vettore dei figli del nodo corrente (Accetta come parametro solo **l'indice numerico**)
.firstChild primo figlio del nodo corrente
.lastChild ultimo figlio del nodo corrente
.parentNode padre del nodo corrente
.previousSibling fratello precedente
.nextSibling prossimo fratello del nodo corrente

Proprietà per l'accesso alle informazioni del nodo

.childNodes.length numero dei figli del nodo corrente
.childElements.count come sopra.. Nel conteggio ignora i white spaces (però non li elimina)
.hasChildNodes() indica se il nodo corrente ha nodi figli
.nodeName legge/imposta il nodeName di un tag
.tagName come sopra
.nodeType legge/imposta il nodeType. Es: if (node.nodeType !== Node.TEXT_NODE)
.nodeValue legge/imposta il nodeValue di un nodo foglia (TextNode)
.textContent contenuto testuale (foglia) di un Element Node. Un nodo con figli è meglio che non abbia TextContent, altrimenti occorre tenerne conto nella navigazione
.innerHTML contenuto testuale completo di un qualunque nodo

Metodi per la gestione degli attributi

```
node.hasAttributes( );
node.hasAttribute(attributeName);
node.setAttribute(name, value);
node.getAttribute(name);
node.removetAttribute(name);
```

Metodi per l'aggiunta / rimozione dei nodi

parentNode.appendChild(childNode) consente di appendere nuovi nodi in coda
parentNode.insertBefore(newNode, childNode) aggiunge newNode davanti a childNode
parentNode.removeChild(childNode) consente di eliminare un nodo

Accesso alla radice dell'albero

```
var root = xmlDoc.documentElement;
var root = xmlDoc.childNodes[0];
var root = xmlDoc.getElementsByTagName("bookstore")[0];
```

Metodi per la creazione di nuovi nodi

xmlDoc.createElement("nodeName")

```
var riga = xmlDoc.createElement("elemento");
root.appendChild(riga);
var foglia = xmlDoc.createTextNode("salve mondo");  // Nodo di tipo foglia
riga.appendChild(foglia);
```

Creazione di un nuovo xmlDoc

var xmlDoc = document.implementation.createDocument("", "", null);

Il primo parametro indica un eventuale namespace da anteporre al documento
Il secondo parametro rappresenta un ulteriore prefisso opzionale
Il terzo parametro indica il tipo di documento (Document Type)

```
var root = xmlDoc.createElement("root");
xmlDoc.appendChild(root);
```

Esempio 1

```
var root = xmlDoc.documentElement;
alert(root.childNodes.length);  // 4

var book = root.childNodes[0];
var title = book.childNodes[0];
alert (title.textContent);  // Everyday Italian

for (var i=0; i<root.childNodes.length;i++){
     var book = root.childNodes[i];
     var category =  book.getAttribute("category");
     var title = book.childNodes[0].textContent;
     var authors =   "";
     for (var j=0; j<book.childNodes.length;j++){
          var field = book.childNodes[j];
          if(field.nodeName == "author")
              authors += field.textContent + "; ";
     }
     alert (title + '\n' + category + '\n' + authors);
}
```

Esempio 2

Si può accedere direttamente ai singoli nodi anche utilizzando i soliti getElementById:

```
var title = xmlDoc.getElementsByTagName("title");
for (var i=0;i<title.length;i++) {
    document.write(title[i].childNodes[0].nodeValue);
    document.write("<br>");
}
```

Creazione di nuovi nodi all'interno della pagina HTML

```
var tab = document.getElementById("gridStudenti");
var riga = document.createElement("tr");
tab.appendChild(riga);
```

Cenni alle Regular Expression in java script

Sintassi Breve

Il pattern di ricerca può essere scritto semplicemente tra due **slash** :

/pattern/modifiers;

```
str = str.replace(/>\s*/g, '>');  // Replace ">  " with ">"
\s = white spaces (tab, line feed, etc)
*  = L'ultimo termine (il white space) può ripetersi un numero imprecisato di volte (anche 0 volte)
```

Principali **`modifiers`** : i = case-insensitive, g = global match (altrimenti si ferma alla 1° occorrenza)

Sintassi Completa

Questa sintassi consente di parametrizzare l'espressione regolare da ricercare.
La stringa passata alla regular expression deve essere preceduta da un \

```
var exp = new RegExp("\>\s*", "g");
str = str.replace(exp, ">");
var exp = new RegExp("\\s*<", "g");
str = str.replace(exp, "<");

\s = white spaces (tab, line feed, etc)
*  = L'ultimo termine (il white space) può ripetersi un numero imprecisato di volte (anche 0 volte)
g  = sostituisce TUTTE le occorrenze
```

Il metodo test()

Il metodo **`test()`** consente di verificare se una stringa soddisfa o meno ad un certo pattern:

```
var patt = /e/;
patt.test("The best things in life are free!");  // return true
```

Vettori Associativi

Sono vettori che al posto dell'indice numerico usano una **chiave** alfanumerica (nell'esempio pippo, pluto e minnie).

```
var vect = new Array(); // oppure var vect=[] raccomandato perchè più veloce
vect['pippp']  = "descrizione di pippo";
vect['pluto']  = "descrizione di pluto";
vect['minnie'] = "descrizione di minnie";
```

I valori dei vettori associativi vengono salvati mediante una tecnica di hash legata alla chiave, per cui il principale vantaggio rispetto ai normali array enumerativi è la **possibilità di accesso diretto tramite chiave** a qualsiasi campo del vettore. Nel caso dei vettori enumerativi, per trovare una informazione occorre eseguire una ricerca (tipicamente sequenziale) all'interno del vettore. Nel caso invece degli array associativi il campo di interesse (es 'minnie') può essere acceduto in modo diretto:

```
alert(vect['minnie']);
```

E' anche possibile, una volta definito il vettore, definire alcune celle tramite chiave, ed altre tramite indice, cioè in pratica creare un vettore misto in cui il ciclo **`for in`** consentirà di scandire le celle associative, mentre il ciclo **`for of`** consetirà di scandire le celle enumerative. I due gruppi sono completamente distinti. Approccio misto comunque assolutamente sconsigliato.

Una oggetto simile in C# sono i Dictionary

Nota

Internamente i vettori associativi possono essere visti come matrici a due colonne del tipo seguente:

```
var vect2 = new Array();
vect2[0] = new Array ('pippo',  'descrizione di pippo');
vect2[1] = new Array ('pluto',  'descrizione di pluto');
vect2[2] = new Array ('minnie', 'descrizione di minnie');
alert(vect2[0][2]);
```

In realtà alla base dei vettori associativi c'è una tecnica di 'hash' non presente nelle matrici per cui scrivere

```
vect1['pippo']  = "descrizione di pippo";
vect2[0] = new Array ('pippo',  'descrizione di pippo');
```

non è esattamente la stessa cosa. Nel primo caso la descrizione viene posizionata in una locazione ben precisa (non necessariamente sequenziale) strettamente dipendente dalla chiave.
Per cui, per quanto simili, facendo `alert(vect2['pippo'])` il risultato sarà undefined,

JSON : Java Script Object Notation

In Java Script i vettori associativi possono essere scritti anche in un modo alternativo molto comodo, cioè utilizzando la cosiddetta Java Script **Object Notation** :

Le righe precedenti avrebbero potuto essere scritte anche nel modo seguente del tutto equivalente :

```
var vect = {
   "pippo" : "descrizione di pippo",
   "pluto" : "descrizione di pluto",
   "minnie" : "descrizione di minnie"
};
```

in cui i vari campi sono scritti nel formato **nome : valore** e separati da una virgola. **Praticamente si definisce un oggetto a partire dal suo contenuto, racchiudendo tra parentesi graffe le sue proprietà ed anche i suoi metodi.** A differenza degli oggetti utilizzati nella programmazione tradizionale, gli Object Jva Script vengono dichiarati ed istanziati allo stesso momento. Non è necessario il new. In realtà la variabile contiene comunque un **riferimento** all'oggetto, esattamente come nella programmazione strutturata.

Terminologia e sintassi

Si chiamano:
- **chiavi** le stringhe che rappresentano il nome della proprietà,
- **valori** gli elementi associati alle chiavi.

In molti casi una struttura come questa è chiamata *hashmap*, ed è caratterizzata da coppie chiave-valore

Le chiavi possono essere scritte indifferentemente
- senza virgolette (cioè come **identificatore**)
- **con le virgolette** (cioè come **stringa**), preferibile ed obbligatorio in jSon
  Le virgolette sono obbligatorie quando il nome non segue le regole per i nomi delle variabili (cioè se contiene spazi o caratteri speciali come ad esempio il trattino: es "Content-Type")

- I valori possono essere scritto come stringhe oppure utilizzando il nome di un'altra variabile.
- **Per le chiavi non è consentito utilizzare il nome di una variabile.** Anche se si omettono le virgolette, il testo viene comunque sempre interpretato come stringa

Esempio : anagrafica di uno studente

```
var studente = {
    "nome" : "mario",
    "cognome" : "rossi",
    "eta" : 16,
    "studente" : true,
    "images" : ["smile.gif", "grim.gif", "frown.gif", "bomb.gif"],
    "hobbies" : [],              // vettore al momento vuoto
    "pos": { x: 40, y: 300 },    // oggetto annidato

    "stampa" :   function () { alert("Hello " + this.nome); },
    "fullName" : function () { return this.nome + " " + this.cognome; }
};
```

Accesso diretto al valore delle proprietà

Si può accedere **direttamente** ai singoli valori in 2 modi:

```
var eta = studente['eta']; // sintassi dei vettori associativi (preferibile)
var eta = studente.eta;    // Object Notation (no PHP)
```

Nel primo caso la chiave di accesso **deve obbligatoriamente** essere scritta come stringa (**con le virgolette**). Lo scopo è quello di consentire un accesso parametrizzato tramite variabile che contiene il nome del campo a cui intendiamo accedere :

```
var myVar = "cognome";
var a = 1;
var nome = studente[myVar];      // "rossi"
var nome = studente["nome"+a];   // accede al campo nome1
```

Se si cerca di accedere in lettura ad una chiave inesistente, il risultato sarà **undefined**
In alternativa si può verificare a monte se la chiave esiste:

```
if(key in studente)    alert(studente[key]);
```

Accesso in scrittura e aggiunta di nuovi record

```
studente['eta']=18;  // sovrascrive il valore precedente
```

Per aggiungere un nuovo record **NON è ammesso l'utilizzo del metodo .push()** (che vale solo per i vettori enumerativi) ma è sufficiente accedere in scrittura ad una chiave inesistente e la nuova chiave verrà aggiunta all'oggetto con il corrispondente valore.

```
studente['indirizzo']="Fossano";     // vale anche per i metodi !
```

Sintassi di Dichiarazione di un Object

```
var persona = new Object();     // sconsigliato
var persona = {};               // più veloce
var persona = new Object({"nome" : "Mario", "cognome" : "Rossi"});
var persona = {"nome" : "Mario", "cognome" : "Rossi"};
var persona = new Array();     // sconsigliato
var persona = [];              // più veloce
```

Attenzione che le ultime due righe non sono esattamente equivalenti alle precedenti. Anche se tutto sembra funzionare allo stesso identico modo, non è possibile ad esempio serializzare la variabile tramite il metodo `.stringify` che restituisce "". In questo caso occorre fare un ciclo come per tutti i vettori.

Scansione delle proprietà di un oggetto

E' possibile scandire il contenuto di un Object tramite il ciclo **for each in**:

```
for (var key in studente)
    alert(key + ' = ' + studente[key]);
```

Notare che `key`, a differenza di C#, non è un oggetto, ma una semplice stringa che contiene il nome della chiave. **Non** è pertanto possibile scrivere una cosa del tipo:

```
alert(key.value)
```

ma occorre scrivere

```
alert(key + ' = ' + studente[key]);
```

Accesso al vettore delle chiavi

```
var keys = Object.keys(studente);
var chiave = keys[0];
var valore = studente[chiave];
```

Riferimento ad un Object

Gli oggetti vengono passati SEMPRE per riferimento. Questo vale anche per le assegnazioni:

```
var studente2 = studente1;     // copia il riferimento
studente2.nome = "enrico";     // anche studente1.nome conterrà "enrico";
```

Utilizzo dei Metodi

Per i metodi occorre scrivere **`methodName: function(){},`** dove methodName rappresenta sostanzialmente un riferimento che punta al metodo. I Metodi, per poter accedere alle Properties della classe, devono **obbligatoriamente** utilizzare la parola chiave **`this`**. Un metodo può invece accedere ad altri metodi anche senza il *this* (esclusi però quelli dichiarati tramite prototype che richiedono il *this*).
In pratica la cosa migliore è quella di utilizzare sempre il this.

Esempi.

```
studente.stampa();       // Hello Mario
alert(studente.fullName());
```

Data una variabile che punta ad un metodo

```
var method =studente['stampa'];
```

sono consentite tutte le seguenti sintassi:

```
alert(method);              // Stampa il codice del metodo, cioè:
                               function() { alert("Hello" + this.name); }
alert(typeof(method));      // function
method();                   // Hello World
studente['stampa']();       // Hello World. Sintassi usata nel dispatch
```

Concatenamento di metodi

Un aspetto interessante e molto sfruttato dei **metodi** degli Object è quello di utilizzare all'interno dei metodi un **return this** finale. Questo fa si che il metodo, dopo aver eseguito le proprie operazioni, ritorni l'oggetto stesso, così che il chiamante possa richiamare in cascata un altro metodo della stessa classe:

```
studente.elabora().stampa().rilascia();
```

Note sugli oggetti js

- Non è possibile visualizzare il contenuto di un object usando un semplice **`alert`**. Gli elementi dell'array associativo sono trattati del motore JavaScript come le proprietà di un oggetto, per cui è necessario scorrere gli elementi uno per uno, usando le rispettive chiavi.
  In realtà alcuni browser, facendo **`alert(studente);`** mostrano tutte le voci, una per riga, con il relativo valore. Però all'interno dell'alert **NON** si possono eseguire concatenamenti, perché altrimenti invece del contenuto viene stampato l'indirizzo a cui punta il riferimento studente

- La proprietà **`length`** di un object è sempre **zero**, per lo stesso motivo di cui sopra: Trattandosi di un oggetto non ha senso parlare della sua lunghezza, intesa come numero di membri

- A differenza delle variabili primitive, gli Object vengono passati alle funzioni per riferimenti. Dunque la sottofunzioni possono apportare modifiche ai valori delle Property.

- Al costruttore Object è possibile passare il risultato di una qualsiasi espressione java script:

  ```
  var N = new Object(72);
  var S = new Object("salve mondo");
  ```

  Queste istanze creano un oggetto a partire da un dato primitive (boxing, esattamente come in java). Object espone due metodi comuni che sono toString() e valueOf() che eseguono sostanzialmente l'unboxing():

  ```
  var s = N.toString();      // "72"
  var n = N.valueOf();       //  72
  ```

Vettori di Object

```
var myArray = [];   // oppure

var myArray = {};
var myArray = new Object();

myArray [0] = {
   'name' : 'Mario',
   'surname' : 'Rossi',
   'age' : '33'
};

myArray [1] = {
   'name' : 'Giuseppe',
   'surname' : 'Verdi',
   'age' : '42'
};

for (i=0; i<myArray.length; i++)
     for (key in myArray [i])
          alert(key + ' = ' + myArray[i][key]);
```

Le specifiche JSON

JavaScript Object Notation
Formato molto utilizzato per lo scambio di dati fra applicazioni client-server, costituito da uno o più Object serializzati. Formato destinato a sostituire XML/XSLT, rispetto al quale è decisamente più leggero. Inoltre

- XML ha una struttura intrinseca gerarchica, dunque è preferibile per dati aventi una struttura gerarchica
- JSON ha invece una struttura tabellare e quindi risulta preferibile per dati aventi struttura tabellare, come lo sono quasi sempre i dati provenienti da un DB (SQL o noSQL).

Specifiche :

- Non è ammesso usare solo numeri come nome di chiave (*ovviamente*)
- I **nomi dei campi** DEVONO essere scritti come stringhe (cioè racchiusi tra doppi apici).
- Per le stringhe (chiavi e valori) NON sono ammessi gli apici singoli ma SOLO i **doppi apici**
  Ad esempio Express accetta SOLO stringhe json scritte in questo modo. Altri ambienti (es jQuery) accettano anche chiavi senza virgolette e ammettono l'utilizzo delle virgolette semplici.
- **Numeri** e **Booleani** possono essere scritti anche in modo diretto (con o senza doppi apici)

Il sito **`jsonformatter`** consente di validare l'esatta sintassi di una qualunque stringa jSon.

Il sito `http://www.httputility.net/`**`json-minifier`**`.aspx` consente interessanti conversioni di dati da XML a JSON e viceversa

Esempi di stringa jSon valide

```
var persona = { "nome" : "pippo" };

var person = {
      "name"     : "Nicolas",
      "age"      : 22,
      "date"     : [01, 09, 1992],   // vettore
      "alive"    : true,
      "gender"   : "Male",
      "student"  : true,
      "info"     : {"web": "myPage.it", "mail": "myMail@vallauri.edu"}
};

var  persone = [
   {
      "name" : "Nicolas",
      "age"  : 22,
   },
   {
      "name"  : "Piero",
      "age"   : 29,
   },
   {
      "name"  : "Gianni",
      "age"  :  20,
   }
];
```

Dati supportati: *http://www.json.org/json-it.html*

`null`
`interi, reali, booleani (true e false)` scritti con o senza i doppi apici. Per i decimali si utilizza il puntino. E' ammessa anche la virgola ma in tal caso occorre utilizzare gli apici doppi
`stringhe` **racchiuse da doppi apici**
`array(`sequenze di valori, separati da virgole e racchiusi in parentesi quadre`);`
`object(`sequenze **coppie** chiave-valore separate da virgole e racchiuse in parentesi graffe`)`
`array of objects` e qualunque altra forma composita

Parsing e Serializzazione di uno stream JSON

Lo standard **ECMA Script 5** che definisce le specifiche di Java Script ha definito un nuovo oggetto **JSON** pienamente supportato dall'engine V8 di google chrome (e quindi anche da Node js).

Il metodo statico **`JSON.parse(str)`** consente di parsificare una stringa jSON convertendola in oggetto
Il metodo statico **`JSON.stringify(obj)`** consente di convertire un oggetto o un vettore di oggetti nella stringa corrispondente.

```
var s1 = '{ "name": "John Doe", "age": 42 }';
var obj = JSON.parse(s1);
alert(obj.name);
var s2 = JSON.stringify(obj);
```

Approfondimenti su Java Script

Un sito di rapido test del codice

www.webtoolkitonline.com

L'operatore ===

Confronta non solo il valore ma anche il tipo

```
var a = 1;
var b = "1";
if (a==b) // true
if (a===b) // false
```

The ternary conditional operator

E' una tecnica disponibile in tutti I linguaggi per compattare al massimo il costrutto if.

Si supponga di dover eseguire le seguenti assegnazioni:

```
if(ok) {
    msg = 'yes';
} else {
    msg = 'no';
}
```

Questo costrutto può essere riscritto in modo molto più compatto nel modo seguente:

```
msg = ok ? 'yes' : 'no';
```

Cioè se la variabile **ok** è vera, allora alla variabile **msg** viene assegnato yes, altrimenti viene assegnato no. Estremamente comodo e compatto.

Note:

1) Attenzione al fatto che, dopo il ?, occorre necessariamente utilizzare o dei valori diretti (come nell'esempio) oppure delle funzioni che restituiscono un valore. Non è consentito utilizzare delle procedure perché non potrebbero assegnane nessun valore a msg
2) msg potrebbe anche essere omesso, nel qual caso il costrutto si limita ad eseguire una delle due funzioni di destra a seconda del valore di ok. Anche in questo caso però a destra non sono ammesse procedure ma sempre soltanto funzioni.