Gestione delle date in Java Script

Rev. 1.3 del 31/05/2024

Siccome il web è utilizzato a livello planetario, l'oggetto javascript **ISODate** o semplicemente **Date** (standard ISO 8601 del 1988) memorizza le date **SEMPRE** con l'ora riferita al meridiano 0 di Greenwich (il cosiddetto **Greenwich Mean Time**). In Italia, che adotta l'ora standard dell'Europa centrale, l'ora solare invernale è +0**1**:00 rispetto al GMT, quella legale estiva +0**2**:00 rispetto al GMT.

- La data viene salvata all'interno dell'oggetto Date come numero intero a **64 bit** che rappresenta il cosiddetto timestamp unix, cioè il numero di **millisecondi** trascorsi dalla data zero dei sistemi unix (1/1/1970).
- Data e ore memorizzate in un oggetto **Date** sono *SEMPRE* riferite al GTM
- I numeri negativi indicano le date antecedenti al 1/1/1970.
- Su 64 bit sono rappresentabili 290 milioni di anni sia in positivo che in negativo.

Creazione di un oggetto Date

Per creare una nuovo **oggetto Date** si utilizza il costruttore dell'oggetto Date che restituisce SEMPRE un **ISODate:**

```
let dataCorrente = new Date();
```

restituisce la data corrente come oggetto **ISODate**.

E' anche possibile creare un oggetto **ISODate** a partire da una generica data scritta come stringa **in formato UTC**:

```
let dataDiNascita = new Date("2020-10-13");
```

I **Formati UTC validi** sono i seguenti (**scritti sempre con il trattino e non con lo slash**)

```
YYYY          viene salvato come  01-01-YYYY ore 00 GMT
YYYY-MM       viene salvato come  01-MM-YYYY ore 00 GMT
YYYY-MM-DD    ore 00 GMT
MM-DD-YYYY    ore 00 GMT
YYYY-MM-DDThh:mm±HH:MM    (L'ora viene interpretata come ora locale)
YYYY-MM-DDThh:mm:ss±HH:MM
YYYY-MM-DDThh:mm:ss.mmm±HH:MM
mmm di un timestamp unix  (Intesi come ora GMT. Es 1651835027598 => 2022-05-05T13:00:00.000Z)
```

```
let dataDiNascita = new Date("2020-10-13");
let dataCompleta = new Date("2020-10-13T00:00:00");
```

Il metodo `Date()` **senza il new** restituisce la data corrente come semplice stringa UTC `YYYY-MM-DD`

Trattino e Slash

- Il **trattino** sta ad indicare una data **GMT** (**Greenwich Mean Time)**
- lo **slash** sta ad indicare invece una data locale

I formati UTC prevedono SEMPRE SOLO il trattino che diventa in pratica sinonimo di GMT

Note sull'impostazione dell'ora (GMT vs Locale)

Se in fase di creazione di una data viene **esplicitamente** indicato anche il il time, allora il time viene interpretato come ora locale. In questo caso la funzione new Date() richiede al sistema operativo il timezone della macchina e, sulla base del timezone, provvede a ricalcolare data e ora come "**timestamp assoluto riferito a Greenwich"** e a memorizzarle come Greenwich time. In qualunque momento, le funzioni di visualizzazioni potranno leggere il timezone dal Sistema Operativo e ricalcolare l'ora locale.

A tale proposito notare la sottile differenza tra i due seguenti gruppi di esempi :

```
let date = new Date("2020-10-13")  //  ora 00 di Greenwich
console.log(date.toISOString())    //  2020-10-13T00:00:00.000Z      //GMT
console.log(date.toLocaleString()) //  13/10/2020, 02:00:00 GMT+0200 //locale
```

```
let date = new Date("2020-10-13T00:00:00")  // ora 00 locale
console.log(date.toISOString())     //  2020-10-12T22:00:00.000Z        //GMT
console.log(date.toLocaleString()) //  13/10/2020, 00:00:00 GMT+0200  //locale
```

Riguardo al ± eventualmente presente in coda alla stringa di formattazione :

- Se omesso, viene richiesto al sistema operativo (come nel secondo caso precedente)
- Se indicato esplicitamente, viene utilizzato per il ricalcolo della data senza richiederlo al sistema operativo.

Principali Metodi di Visualizzazione di un oggetto Date

```
let dataCorrente = new Date("2020-10-13T17:37:56");
```

dataCorrente.**toString()** Tue Oct 13 2020 **17:37:56** GMT+0200 // stringa locale
Restituisce data e ora locali. Rappresenta il metodo di default utilizzato in fase di serializzazione
La funzione richiede il timezone locale al sistema operative, lo somma alla data contenuta all'interno dell'oggetto Date e la visualizza come data / ora locali.

dataCorrente.**toUTCString**() (Tue, 13 Oct 2020 15:37:56 GMT // stringa GMT
Simile alla precedente ma restituisce data e ora GMT

dataCorrente.**toISOString**() **2020-10-13T15:37:56.000Z** // UTC standard
Restituisce la data nel cosiddetto **formato UTC standard** costituito da **24 caratteri** **`YYYY-MM-DDThh:mm:ss.mmmZ`** con uno **Z** finale che indica che l'ora visualizzata è riferita al meridiano 0 di Greenwich. E' l'unico formato accettato dal BSON $date di mongodb per salvare una data all'interno di un DB. Concatenando .**substring(0,10)** si ottiene la tipica data UTC da poter passare ad un `<input type=date>` Non esiste un metodo corrispondente .toISODateString()

dataCorrente.**toLocaleString()** 13/10/2020, **17:37:56** // data/ora locali
dataCorrente.**toLocaleTimeString**() **5:37:56** // solo ora locale
dataCorrente.**toLocaleDateString**() 13/10/2020 // solo data locale
Queste date **NON** possono essere passate al costruttore perché, essendo locali, usano come separatore lo slash e non il trattino. Le ultime due accettano come parametro dei comodissimi formattatori. Esempi:

```
let timeOptions = {'it-IT', { hour12: false }}
dataCorrente.toLocaleTimeString(timeOptions)  17:37:56
let dateOptions = {'it-IT', {"year":"numeric",  "month":"long",  "day":"numeric"}}
dataCorrente.toLocaleDateString(dateOptions)  13 ottobre 2020
```

dataCorrente.**getTimezoneOffset**() Restituisce l'offset (in minuti) del meridiano corrente (+60 per l'Italia)

Metodi che restituiscono il timestamp interno

msec=dataCorrente.**getTime( )** Restituisce il **timestamp unix interno** (msec dal 1/1/70)**)**
msec=dataCorrente.**valueOf( )** equivalente al precedente
msec=**Date.now( )** Restituisce la data corrente in msec

Accesso ai singoli campi di una data

ss = dataCorrente.getSeconds ( ) Restituisce i secondi 0-59
mm = dataCorrente.getMinutes( ) Restituisce i minuti 0-59
hh = dataCorrente.getHours( ) Restituisce l'ora da 0-23 riferita al timeZone locale
gg = dataCorrente.**getDate( )** Restituisce il giorno come numero intero da **1-31** riferita al timeZone locale
mth = dataCorrente.getMonth( ) Mese dell'anno 0-11 (attenzione gennaio = 0)
yy = dataCorrente.getFullYear( ) Anno a 4 cifre

gg = dataCorrente.getDay ( ) Giorno della settimana 0-6 (Domenica = 0, Lunedì = 1, …… Sabato = 6)

hh = dataCorrente.getUTCHours( ) Restituisce l'ora GMT da 0-23
gg = dataCorrente.getUTCDate( ) Restituisce il giorno come numero intero da **1-31**espresso in GMT
gg = dataCorrente.getUTCDay ( ) Giorno della settimana 0-6 (Domenica = 0, Lunedì = 1, …… Sabato = 6) UTC
mth = dataCorrente.getUTCMonth( ) Mese dell'anno in GMT (0-11)

Modifica di una data

I seguenti metodi non restituiscono alcunché, ma provvedono semplicemente a **modificare il contenuto di date**:
dataCorrente.setTime(val) Si aspetta come parametro i msec complessivi e reimposta l'intera data
dataCorrente.setSeconds (val) Modifica soltanto i secondi
dataCorrente.setMinutes(val) Modifica soltanto i Minuti
dataCorrente.setHours(val) Modifica soltanto l'ora (interpretata come locale)
dataCorrente.**setDate(val)** Assegna il giorno del mese 1-31. Per aggiungere un giorno alla data corrente:
`date.setDate(date.getDate() + 1)` //Aggiorna automaticamente anche **mese** ed **anno**.
dataCorrente.setMonth(val) mese dell'anno 0-11
dataCorrente.setYear(val) Anno a 4 cifre
dataCorrente.setDay(val) Modifica soltanto il giorno della settimana 0-6

dataCorrente.setUTCHours(val)Modifica soltanto l'ora (interpretata come UTC)
dataCorrente.**setUTCDate**(val) Assegna il giorno del mese 1-31 calcolato come giorno UTC
dataCorrente.setUTCDay(val) Modifica soltanto il giorno della settimana 0-6 (UTC)
dataCorrente.setUTCMonth(val) mese dell'anno 0-11

Assegnazione di una data ad un `input[type=date]`

Per assegnare il valore ad un **`input[type=date]`** occorre creare manualmente una stringa **in formato UTC** del tipo : YYYY-MM-GG che può essere costruita manualmente nel modo seguente:

```
let year = date.getFullYear()
let month = ("0" + (date.getMonth() + 1)).slice(-2);
let  day = ("0" + date.getDate()).slice(-2);   // slice(-2) restituisce gli ultimi due chr
let s = year + "-" + month + "-" + day
```

Oppure più velocemente facendo uso di uno dei seguenti metodi:

```
input.value = dataCorrente.toISOString().substring(0,10)   // oppure
input.valueAsDate = new Date()
```

Somma, sottrazione e confronto fra date

Poiché il timestamp è un semplice numero intero, **è possibile sommare e sottrarre i timestamp.**

- Il **confronto** può sempre essere eseguito in forma diretta: `if(data1> data2) …`
- Nel caso della **sottrazione** può essere eseguita direttamente sull'oggetto ISODate, **senza** l'utilizzo di .getTime().
- Nel caso della **somma** occorre sempre passare attraverso .getTime()

1) Data che si avrà fra 7 giorni.

```
let dataCorrente = new Date()
let nextWeek = dataCorrente.getTime()    +    7 * 24 * 3600 * 1000
alert(new Date(nextWeek))
```

2) Differenza fra due date

La differenza fra due oggetti ISODate **restituisce i millisecondi intercorsi fra le due date**.
Dividendo per 1000 trovo i secondi, Dividendo per 1000 poi per 60 trovo i minuti.

```
let diff = new Date(_txtN2.value) - new Date(_txtN1.value)
let nGiorni = Math.floor(diff/1000/60/60/24)
```

Se la differenza da valutare è inferiore al mese si può utilizzare la seguente soluzione.

```
let dateDiff = new Date(diff);
let giorni = dateDiff.getUTCDate();  let hours = dateDiff.getUTCHours()
let minutes = dateDiff.getUTCMinutes() // senza UTC verrebbe aggiunto il timeZone
```

3) Età di una persona

```
let age=Math.floor((new Date() - dob)/(365*24*3600*1000))
```

4) Come impostare una data 'locale' partendo da una data UTC

```
let offset = dataCorrente.getTimezoneOffset() / 60;
let hours = dataCorrente.getHours();
dataCorrente.setHours(hours - offset);
```

Nota

Se si imposta una certa data tramite new Date() e poi si vanno a leggere i msec contenuti all'interno della data e li si visualizza all'interno di un time converter, in realtà quello che leggeremo sarò il valore GMT cioè, in inverno, si vede un'ora in meno rispetto all'ora italiana utilizzata nel costruttore (due ore in estate).

Cenni sulla libreria moment.js

E' un oggetto simile all'oggetto base ISODate di javascript che però al suo interno memorizza più informazioni sulla data memorizzata, in modo da semplificare notevolmente le elaborazioni sulle date, in particolare somme, sottrazioni, differenze mediante i comodissimi metodi **.add()** **.subtract()** e **.diff()**

Il costruttore moment()

Funzionamento analogo al **`new Date()`** di javascript.
Nel caso di moment() **non** si utilizza il **new** che viene eseguito all'interno della libreria

`let date = moment()` restituisce un oggetto moment contenente la data corrente
`let date=moment("2020-10-13")` restituisce un oggetto moment riferito alla data **`2020-10-13`**
`let date=moment("2020-10-13T16:00:00")` restituisce un oggetto riferito a data e ora indicati.
`let date=moment("16:00:00", "HH:mm:ss")` come data viene impostata la data corrente.
Senza il secondo parametro l'istruzione andrebbe in errore.

L'ora viene SEMPRE interpretata come **ora locale** (esattamente come in javascript) a meno che non sia indicata esplicitamente la **Z** finale che indica un Greenwich Time. A differenza però di quanto accade in javascript, se si omette l'ora, essa viene assunta come **ora 0 locale** e non GMT (Greenwich Mean Time)

Come parametro si possono passare anche i **msec** (provenienti ad esempio da una differenza).
In quest'unico caso il valore viene interpretato come **ora Greenwich** del 1/1/70, che potrà poi essere visualizzata come ora GMT (Greenwich Mean Time) oppure come ora locale.

La visualizzazione della data come stringa

Gli oggetti moment dispongono del metodo **format()** che per default restituisce **l'ora locale,** cioè richiede al SO il timezone locale e lo aggiunge all'ora GMT.

```
console.log(date.format())          //2020-10-13T17:37:56+02:00
console.log(date.format("H"))             // 17
console.log(date.format("DD/MM/YYYY"))    // 13/10/2020
console.log(date.format("HH:mm:ss"))      // 17:37:56    H=24ore,  h=12ore
```

Eventuali espressioni tra parentesi quadre vengono visualizzate così come sono.

```
console.log(date.format("[ore] h [e] m [minuti]"))   // ore 5 e 37 minuti
```

All'interno del metodo format() sono utilizzabili tutti gli **shorthand** indicati nella tabella della pag successiva

Nota: Il formato restituito da format è estremamente comodo, però senza lo **Z** finale e con lunghezza **> 24.**
In tal caso mongo non lo considera un formato UTC standard e pertanto non lo accetta per l'inserimento in un DB.

I metodi di istanza .utc() e .local()

Questi metodi "agiscono" su un campo interno alla data memorizzata nell'oggetto moment.
Definiscono in pratica come dovrà comportarsi il metodo `format()`.
`data1.utc()` fa sì che i successivi metodi format() su data1 restituiscano data/ora in GMT
`data1.local()` fa sì che i successivi metodi format() restituiscano l'ora in formato locale.

Le modalità **`utc()`** e **`local()`** possono essere applicate anche direttamente in fase di creazione dell'oggetto moment():

```
let date=moment.utc("2020-10-13T16:00:00")
let date=moment.local("2020-10-13T16:00:00")  // default
```

Il metodo di istanza .unix()

Restituisce il timestamp interno inteso come numero di msec trascorsi dal 1/1/70 con riferimento GMT.
I vari timestamp converter presenti in rete visualizzano la corrispondente data/ora GMT e, dopo aver richiesto il timezone al sistema operativo, la corrispondente data/ora locale.

```
data1.unix()
```

Il risultato restituito dal metodo .unix() è *sempre* GMT; **indipendentemente** dall'utilizzo di .utc() e .local() che agiscono SOLO ed ESCLUSIVAMENTE sul metodo format()

Funzioni di estrazione diretta delle informazioni contenute in una data 'moment'

Su un oggetto moment sono disponibili tutte le seguenti funzioni che ritornano l'informazione richiesta.

- A destra sono riportati gli shorthand corrispondenti utilizzabili all'interno del metodo format()
- Tutti i metodi possono essere scritti indifferentemente in forma singolare oppure plurale (cioè la **`s`** finale è facoltativa)
- Se usati senza parametro restituiscono il valore richiesto
- Se viene passato un parametro, questo viene utilizzato in assegnazione

Key	**Shorthand [to use in format() method]**
years	**Y** - **YY** - **YYYY**
quarters	Q
months	**M** - **MM** (zero padded) a base 0 - **MMM** Jan Feb ... Nov Dec
weeks	W
days	**D** - **DD** (zero padded)
day of week	d - ddd (Sun Mon ... Fri Sat)
hours	h - hh (zero padded)
minutes	m - mm (zero padded)
seconds	s - ss (zero padded)
milliseconds	ms

```
let anno = data1.year()
let ora = data1.hour()              // ora contenuta in data1
let newDate = moment().seconds(0)   // azzero i secondi del moment corrente
```

Tabella completa di tutti i formattatori:
https://momentjscom.readthedocs.io/en/latest/moment/04-displaying/01-format/

Il confronto

Le date di moment, così come le ISODate, possono sempre essere confrontate in modo diretto:

```
if(data1 > data2) …
```

I metod add() e subtract()

I metodi .add() e .subtract() applicati ad un oggetto moment consentono una rapida elaborazione

```
let date = moment().subtract(1, 'year')
let date = moment().subtract(1, 'day')
let date = moment().add(1, 'day')
```

```
moment().add(7, 'days').add(1, 'months');
moment().add( {days:7, months:1} )
moment().add(7,'days').subtract(1,'months').year(2009).minutes(0).seconds(0)
```

Attenzione che all'interno dei metodi **add(), subtract() e diff()** NON bisogna utilizzare gli shorthand ma i nomi dei metodi !!

Il metodo diff()

Restitusice la differenza in msec fra due "moment Date" (esattamente come la differenza diretta fra le ISODate). Espone però la possibilità di specificare un secondo parametro che indica come deve essere restituito il risultato. Accetta come parametri gli stessi di cui sopra: `days, hours, minutes, etc.`

```
// Differenza fra le due date espressa come numero di giorni
let ggPermanenza=dataFine.diff(dataInizio, "days")
```

Se si omette il secondo parametro, il risultato viene restituito come numero di **millisecondi** (intesi come orario di Greemwich del 1/1/70), che possono essere riconvertiti in oggetto moment passandoli come parametro al costruttore moment().

Attenzione però che se si vogliono eseguire delle elaborazioni sul risultato espresso in msec, occorre abilitare la modalità **`utc()`** perché se viene utilizzata la modalità **`local()`**, essa aggiunge una o due ore al numero dei millisecondi:

```
let diff=dataFine.diff(dataInizio)
let ore = moment(diff).utc().hours()   // oppure

let ore = moment.utc(diff).hours()
```

il metodo toDate() e la conversione in ISODate

Dato un oggetto moment, è possibile convertirlo in ISODate usando il metodo .toDate()

```
let date = moment().subtract(1, 'year')
let jsdate = date.toDate()
```

I metodi timeTo() e timeFrom()

Consentono di calcolare in modo molto fine la differenza fra due date.
Per l'utilizzo si rimanda alla sezione DOCS del sito ufficiale