Guida jQuery

di: Marco Solazzi

Introduzione

I selettori

Attributi, classi e metodi base

Traversing

Manipolazione del DOM e dei CSS

Eventi

16. Associare gli eventi: bind e unbind
Associare e rimuovere gli handler

17. Metodi speciali e 'event delegation'
Una panoramica su altri metodi specifici legati agli eventi

Animazioni

18. La coda degli effetti
capire il meccanismo fondamentale che regola l'esecuzione di animazioni

19. Effetti di base
Una panoramica sintetica sugli effetti di animazione di base offerti dal framework

20. Animazioni personalizzate
Come creare effetti personalizzati

Ajax

21. Eventi locali e globali
Panoramica dei principali eventi Ajax gestiti da jQuery

22. jQuery Ajax
I metodi e i parametri fondamentali per gestire chiamate Ajax

23. Scorciatoie e funzioni di supporto
Utili scorciatoie per alcuni tipi di richieste molto utilizzate

Utility

24. Lavorare con gli array
Analizziamo questa fondamentale utilità offerta dal framework

25. Lavorare con gli oggetti
Serializzazione degli elementi, stringhe e funzioni

26. Browser e Feature Detection
Determinare quale sia il browser usato dall'utente e le sue funzionalità

27. Internals e Data storage
Funzioni interne e raccolta dei dati

Estedere JQuery

28. Aggiungere selettori
Analisi di uno dei metodi per estendere facilmente le funzionalità della libreria

29. Aggiungere metodi
Una via alternativa per estendere il framework

30. Usare gli alias
Risolvere conflitti con altri framework

Introduzione

A guardare bene, sotto lo strato della socialità, dell'interattività e dello *User Generated Content*, il web 2.0 con il quale tutti noi abbiamo a che fare giornalmente è basato massicciamente sull'uso di JavaScript. In passato spesso associato all'idea di popup invasivi e fraudolenti, questo linguaggio è stato particolarmente rivalutato negli ultimi anni anche grazie alla diffusione di tecniche come AJAX e alla possibilità sempre più estesa di riprodurre effetti grafici dinamici senza la necessità di plugin come Flash.

Dal punto di vista degli sviluppatori, tuttavia, la natura *lato client* del linguaggio è da sempre la fonte principale di problemi e mal di testa. Infatti, poiché ogni browser implementa uno specifico motore JavaScript, ognuno (ma soprattutto IE) con specifiche ed eccezioni proprie, è spesso impossibile essere certi del funzionamento cross-browser di uno script. Inoltre, il linguaggio presenta ancora notevoli lacune che lo rendono incoerente rispetto ai linguaggi server-side come PHP o Ruby.

In risposta a questi problemi sono nati progetti di librerie (o meglio *framework*) in grado di garantire il funzionamento cross-browser degli script e di estendere o comunque facilitare le funzioni native di JavaScript.

Uno dei primi framework a vedere la luce è stato **Prototype** (http://www.prototypejs.org/), che ha approcciato il problema estendendo il DOM e gli oggetti globali di JavaScript, cioè aggiungendo funzionalità agli elementi che compongono la pagina e ad oggetti come array e strighe. Con questo metodo vengono colmate le numerose lacune di JavaScript allo stesso modo in cui vengono aggiunti plugin in vari programmi. Nelle sue prime versioni anche **Mootools** (http://mootools.net/) aveva intrapreso questa strada, mentre nell'ultima versione la libreria è quasi completamente strutturata ad oggetti, mantenendo l'estensione di alcuni oggetti fondamentali come array e stringhe. Il problema dell'approccio di Prototype è infatti che estendendo gli oggetti globali di JavaScript si potrebbero creare **incompatibilità con nuove versioni dei motori** o addirittura conflitti con le funzioni dello sviluppatore o con altre librerie.

Il progetto jQuery

Un approccio molto diverso è da sempre quello di **jQuery** (http://jquery.com/), un framework sviluppato da John Resig a partire dal 2006 con il preciso intento di rendere il codice più sintetico e di limitare al minimo l'estensione degli oggetti globali per ottenere la massima compatibilità con altre librerie.

Da questo principio è nata una libreria in grado di offrire un'ampia gamma di funzionalità, che vanno dalla manipolazione degli stili CSS e degli elementi HTML, agli effetti grafici per passare a comodi metodi per chiamate AJAX cross-browser. Il tutto, appunto, senza toccare nessuno degli oggetti nativi JavaScript.

Per chi si avvicina a jQuery per la prima volta è quindi fondamentale sapere che tutto ruota attorno all'oggetto/funzione $, a sua volta un abbreviazione (o alias) di `jQuery`.

```
$("#mioBlocco"); //Un oggetto jQuery
jQuery("#mioBlocco") //Lo stesso oggetto richiamato in modo diverso
```

Ma andiamo con ordine, ecco un primo esempio per avere un colpo d'occhio su vantaggi di jQuery:

```
// trovare il valore dell'attributo href del tag a con id mioLink
// <a id="mioLink" href="http://www.html.it">link</a>

document.getElementById("mioLink").href; // JavaScript nativo
$('mioLink').readAttribute('href'); // Prototype
document.getElementById("mioLink").readAttribute('href') // Prototype, come detto, estende il DOM
$("#mioLink").attr("href"); // jQuery
```

Già da queste righe di codice si intuiscono due caratteristiche fondamentali di jQuery: anzitutto la **brevità del codice utilizzato**, ma soprattutto il fatto che l'elemento da ricercare sia stato passato alla funzione `$()` sotto forma di selettore CSS (come avviene in Mootools per la funzione $$).

Per chi non ha dimestichezza con framework del genere, o proviene da altre librerie, inizialmente questa caratteristica potrebbe sembrare uno svantaggio e magari produrrà qualche errore: per esempio in jQuery la funzione `$('mioLink')` cercherà un immaginario tag `mioLink` senza trovarlo.

In realtà uno dei punti di forza del framework è proprio il potente e performante motore di selezione (di cui parleremo più avanti).

Un'altra caratteristica che va sottolineata è che molti metodi di jQuery sono **concatenabili**, per rendere la scrittura e la lettura del codice molto più lineare:

```
$("#mioLink").text("Nuovo testo").css("color","red");
//cambio il testo del link e il colore
```

Esempio (http://www.html.it/guide/esempi/jquery/esempi/introduzione.html).

I buoni motivi per usare jQuery

Oltre ai vantaggi già citati, vi sono alcuni buoni motivi per cui è una buona idea usare jQuery: anzitutto perché è **possibile usarla in tutti progetti** senza paura di incappare in incompatibilità nel codice. Infatti, utilizzando la funzione `jQuery.noConflict()` verrà semplicemente rimosso l'alias $, e potremo usare, per esempio, Mootools richiamando questo framewok con `$('mioId')` e jQuery con `jQuery("#mioId")` (esempio (http://www.html.it/guide/esempi/jquery/esempi/introduzione2.html)).

In secondo luogo, jQuery ha un semplice **sistema di estensione** che permette di aggiungere nuove funzionalità (plugin) oltre a quelle predefinite.

Inoltre la sua diffusione ha fatto in modo che attorno al team di sviluppo ufficiale crescesse una **numerosa community** che mette a disposizione plugin e supporto di ottimo livello.

Infine, e potrebbe sembrare una cosa da poco, perché il motto di jQuery è *"Write less, do more"*, ed effettivamente la sua **sintassi sintetica ed efficiente** è particolarmente apprezzabile quando si tratta di scrivere svariate linee di codice.

Ottenere jQuery

Dal sito **jquery.com** (http://jquery.com/) è possibile scaricare il framework in due formati:

- **development**: non compresso, utile in fase di sviluppo e debug (120Kb)
- **production**: compresso, da utilizzare quando il progetto è online per ottimizzare i tempi di caricamento (59Kb, 19Kb se compresso ulteriormente lato server con GZip)

Una volta scaricato il file, sarà sufficiente collegarlo alla pagina HTML inserendo nel tag head il seguente codice:

```
<script language="javascript" type="text/javascript" src="jquery-1.3.2.min.js"></script>
```

Un'altra soluzione che si è fatta strada nell'ultimo periodo, è quella di utilizzare una copia del framework presente sul **network di Google**:

```
<script language="javascript" type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

Questa soluzione lega il funzionamento dei nostri script alla raggiungibilità del servizio (mai garantita al 100%) ma ha il vantaggio che l'utente potrebbe già aver scaricato il file in questione nella cache del browser e per questo gli verrebbe evitato di dover riscaricare un file uguale con conseguente **diminuzione del tempo di caricamento della pagina**.

Versione originale: http://javascript.html.it/guide/lezione/4142/introduzione/

Per un pugno di $: il motore di jQuery

Come detto in precedenza tutto ciò che possiamo fare con jQuery gira intorno all'oggetto/funzione $, il cui utilizzo più comune è quello di selettore CSS:

```
var elementi = $("#mioId");
```

In questo esempio la variabile `elementi` rappresenta un oggetto jQuery contentente un riferimento all'elemento con id `mioId`.
Come detto in precedenza la stringa passata alla funzione deve rappresentare il **selettore CSS degli elementi da ricercare**, siano essi un singolo id oppure elementi accomunati da una classe:

```
$(".miaClasse"); //tutti gli elementi con attributo class "miaClasse"
```

Come in un foglio di stile sarà possibile indicare **più di un selettore** anche con sintassi piuttosto complesse:

```
$(".miaClasse, #mioId, ul#mioMenu li");
//cerca una classe, un id e gli elementi di una specifica lista puntata
```

Vedremo più avanti le possibili combinazioni di selettori CSS ricercabili con jQuery, ma per completezza è importante dire che, nel caso di ricerca con selettori CSS, è possibile passare alla funzione un secondo argomento per indicare un contesto nel quale operare:

```
$(".miaClasse",document.getElementById('mioId'));
//Tutti gli elementi con classe "miaClasse" dentro l'elemento DOM con id mioId
```

Questa caratteristica sarà molto utile quando dovremo ciclare fra gli elementi raccolti da $ per lavorare direttamente su ognuno di loro.

Inutile dire che $ accetta come primo argomento anche riferimenti diretti ad elementi del DOM:

```
$(document.getElementById('mioId'));
$(window);
```

Creare nuovi elementi

Proprio per rispettare la sinteticità del progetto la stessa funzione $ può essere utilizzata anche per altri scopi, che verranno interpretati dalla funzione in base agli argomenti passati. Così è possibile creare un nuovo elemento nel DOM semplicemente passandone il tag:

```
$("<div></div>");
```

o addirittura:

```
$("<div/>");
```

In questo modo possiamo anche creare elementi complessi in un'unica stringa di codice:

```
var blocco = $("<div><p>Esempio di elementi <strong>nidificati</strong></p></div>");
```

Comunque avremo la possibilità di applicare tutti i metodi di jQuery ai nuovi elementi:

```
blocco.addClass("nuovaClasse"); //aggiungo una classe al div
```

Infine, per inserire questi elementi all'interno della pagina useremo, per esempio, il metodo appendTo:

```
blocco.appendTo(document.body); //inserisci nel body
blocco.appendTo("#contenitore"); //inserisci dentro un altro elemento
```

Ecco un esempio (http://www.html.it/guide/esempi/jquery/esempi/core-1.html) pratico.

Lanciare funzioni al caricamento del DOM

Una caratteristica particolare di $ è quella che permette di passare come argomento una funzione. In questo caso le istruzioni passate verranno lanciate "*on DOM ready*", cioè quando l'albero degli elementi HTML è stato caricato:

```
$(function () {
     alert("DOM caricato!");
});
```

Questo script differisce da:

```
window.onload = function () {
     alert("Caricato!");
};
```

in quanto nel secondo la funzione viene lanciata quando tutto il documento è stato caricato, compresi i fogli di stile e le immagini. L'evento `DOM ready`, invece, permette di associare funzioni agli elementi del documento con la sicurezza che essi verranno eseguiti anche se l'utente li richiama prima del caricamento completo della pagina (esempio (http://www.html.it/guide/esempi/jquery/esempi/core-2-domready.html)).

Versione originale: http://javascript.html.it/guide/lezione/4143/per-un-pugno-di-il-motore-di-jquery/

Selettori di base e gerarchici

Come visto nella lezione precedente, uno dei punti di forza di jQuery è il suo potente motore di selezione CSS. Il suo compito è di selezionare specifici elementi all'interno di un documento per poterci lavorare successivamente con i metodi jQuery.

A partire dalla versione 1.3 il creatore di jQuery, John Resig, ha deciso di riscrivere e rendere indipendente il codice che ora è disponibile come progetto standalone. Secondo Resig, il motore **Sizzle** (http://sizzlejs.com/), questo il nome, potrà crescere indipendentemente da jQuery e magari contare su apporti di altri team per migliorare le sue già ottime prestazioni.

Ma passiamo alla pratica: come abbiamo già visto in precedenza, per selezionare degli elementi nel documento sarà sufficiente identificarli attraverso un **selettore CSS** (lo stesso che usereste in un foglio di stile) e passare la stringa alla funzione $:

```
$("ul#menu li") //ora nell'oggetto jQuery saranno disponibili tutti gli elementi lista del menu
```

jQuery, oltre a **tutta la gamma di selettori CSS3**, comprende alcuni filtri speciali in grado di semplificarci molto la vita.

Selettori di base

Per prima cosa potete utilizzare **selettori semplici** (tutte le combinazioni di tag, classi e id):

```
$("a"); //tutti i link nel documento
$("#blocco"); //seleziona un singolo elemento con id "blocco"
$("a.external"); //solo i link con classe "external"
```

Come in una regola di un foglio di stile CSS, possiamo indicare più selettori contemporaneamente:

```
$("#blocco,a.external");
```

Infine è possibile selezionare tutti gli elementi di un documento:

```
$("*");
```

In quest'ultimo caso, ricordiamo sempre che maggiore sarà la precisione con cui identificheremo gli elementi che ci servono, maggiore sarà la velocità con cui jQuery li raccoglierà ma soprattutto minore sarà il peso dello script per il browser.

Selettori gerarchici

Un'altra possibilità è quella di utilizzare i **selettori gerarchici** per ricercare gli elementi all'interno di altri elementi piuttosto che all'interno di tutto il documento. Il selettore gerarchico più conosciuto è senza dubbio quello ***padre > figlio***:

```
$("#blocco a"); //i link all'interno dell'elemento con id "blocco"
```

Tuttavia jQuery supporta altri selettori che possono risultare spesso molto utili come ***padre > figlio***, che permette di selezionare solo gli elementi che sono direttamante contenuti nell'elemento padre:

```
$("ul > li"); //solo gli elementi lista diretti discendenti del tag ul
```

Ecco un esempio pratico (http://www.html.it/guide/esempi/jquery/esempi/selettori-1.html) per capire la differenza con il selettore precedente

Altri due selettori interessanti sono ***precedente + seguente***, in cui vengono selezionati tutti gli elementi preceduti direttamente da uno specifico elemento:

```
$("label + input"); //tutti i tag input preceduti da un tag label
```

oppure ***precedente ~ adiacente***, che seleziona tutti gli elementi direttamente adiacenti ad un elemento (esempio (http://www.html.it/guide/esempi/jquery/esempi/selettori-2.html)):

```
$("#prev ~ div");
```

Versione originale: http://javascript.html.it/guide/lezione/4147/selettori-di-base-e-gerarchici/

Selezionare per attributi

Un metodo di selezione degli elementi molto utile è quello che permette di indicare regole specifiche per gli **attributi degli elementi**. Come in precedenza, la sintassi non si discosta da quella dei CSS:

```
$("a[target='_blank']"); //tutti i link che si aprono in nuove finestre
```

In questo primo esempio abbiamo individuato un valore ben definito per l'attributo `target`. Tuttavia possiamo indicare anche solo una parte del valore:

```
$("a[title^='nuova']"); //link con titolo che inizia per "nuova"
$("a[title$='pagina']"); //link con titolo che finisce per "pagina"
```

Oppure ancora escludere dalla selezione elementi con determinati valori:

```
"$("a[target!='_blank']"); //link che non aprano una nuova pagina del browser
```

In realtà possiamo decidere di ricercare tutti gli elementi con uno specifico attributo senza tener conto del suo valore. Quest'ultimo selettore potrebbe ritornarci utile nel cso volessimo sostituire il tooltip di default del browser con uno script più complesso come tooltip (http://bassistance.de/jquery-plugins/jquery-plugin-tooltip/):

```
$("a[title]").tooltip(); //tutti i link che hanno impostato l'attributo "title" diventano un tooltip grafico
```

Inutile dire che è possibile concatenare i selettori di attributi per raffinare al meglio la nostra ricerca:

```
$("a[target='_blank'][title='nuova pagina']");
```

In questa pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/selettori-3.html) potete testare i selettori di attributi con alcuni interessanti scenari che vi si potrebbero prospettare in fase di sviluppo.

Versione originale: http://javascript.html.it/guide/lezione/4148/selezionare-per-attributi/

Filtrare le selezioni

I filtri del motore di selezione si presentano con una sintassi molto simile aglii pseudoselettori CSS, cioè preceduti dai due punti ":".

Attraverso questi selettori speciali (spesso un'estensione dei selettori nativi CSS) è possibile non solo raffinare la selezione degli elementi, ma raggruppare molto velocemente degli elementi specifici.

Il primo gruppo di filtri riguarda gli "*elementi figli*", ed è in grado di fornire un metodo di selezione degli elementi in base alla loro posizione rispetto all'elemento contenitore. Nell'esempio qui sotto, il primo selettore troverà solo il primo elemento della lista con id menu, mentre per selezionare l'ultimo useremo il secondo:

```
$("#menu li:first-child");
$("#menu li:last-child");
```

Infine per ricercare solo gli elementi singoli utilizzeremo:

```
$("#menu li:only-child");
```

Un caso più complesso è quello del filtro `:nth-child()` che accetta, tra le parentesi, varie opzioni:

- even: solo elementi pari
- odd: solo elementi dispari
- `indice`: un numero che indichi la posizione (pertendo da 1)
- `equazione`: un'equazione

Ecco una pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/selettori-4.html) sui possibili utilizzi di questi filtri.

Filtri per gli elementi ed il contenuto

In questa categoria si trovano tutti quei selettori per **filtrare gli elementi di una pagina** in base a specifici criteri come per esempio la visibilità:

```
$("div:hidden"); //tutti i tag div nascosti
```

o ancora in base al loro contenuti:

```
$("#menu li:has(ul)"); //tutte voci di menu con sottomenu
```

o, in negativo, in base a ciò che non sono:

```
$("a:not(.external)"); //tutti i link eccetto quelli con classe external
```

Anche per questo tipo di filtri non manca la possibilità di filtrare gli elementi in base alla loro posizione nel documento. In questo caso, però, non viene tenuto conto della loro posizione relativamente agli altri elementi adiacenti, bensì vengono presi in considerazione tutti gli elementi anche se ad un livello superiore (esempio (http://www.html.it/guide/esempi/jquery/esempi/selettori-5.html)):

```
$("#menu li:first"); //il primo tag lista
$("#menu li:odd"); //tutti gli elementi dispari di una lista
$("#menu li:eq(3)"); // il terzo elemento di una lista
```

Filtri per i form

Un gruppo di filtri molto interessante è quello dei form. Permette infatti di lavorare con molta facilità sugli elementi dei moduli, un esempio:

```
$("input[type='text']");
```

può essere scritto anche:

```
$(":text");
```

Con la stessa brevità possiamo ricercare elementi gli elementi disabilitati, magari per abilitarli dinamicamente:

```
$(":disabled").removeAttr("disabled");
```

o ancora trovare velocemente i checkbox selezionati dall'utente:

```
$(":checked");
```

Per capire meglio l'utlità di questi filtri guardate la pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/selettori-6.html).

Riferimenti:

- Documentazione sui selettori (http://docs.jquery.com/Selectors)
- Showcase dei selettori jQuery (http://codylindley.com/jqueryselectors/)

Versione originale: http://javascript.html.it/guide/lezione/4149/filtrare-le-selezioni/

Metodi di base

Nelle lezioni precedenti abbiamo visto come sia possibile "raccogliere" specifici elementi da un documento HTML attraverso il motore di selezione CSS.

Una volta definita la collezione di elementi sulla quale lavorare possiamo applicarvi i metodi jQuery necessari per manipolare un elemento, un attributo o raffinare ulteriormente la nostra selezione. In questa lezione vedremo alcuni metodi fondamentali per iniziare a lavorare con jQuery.

Dimensione della selezione

Anzitutto è possibile scoprire molto facilmente quanti elementi sono stati trovati dal motore di selezione richiamando il metodo `.size()` o addirittura la proprietà `.length` (tipica degli array), consigliata per la maggiore velocità di risposta:

```
$(#menu li).size();
$(#menu li).length;
```

Ciò che viene restituito è un numero che potremo utilizzare, per esempio, all'interno di strutture di controllo:

```
var liste = $("#menu li");
if (liste.length > 0 ) {
                // esegui del codice solo se ci sono elementi
}
```

Estrazione e indice degli elementi

Una volta verificata la presenza di elementi all'interno dell'oggetto jQuery, potremmo voler lavorare direttamente su alcuni di essi richiamando il metodo `.get()` che restituirà una collezione di elementi che potremo elaborare con i metodi nativi di JavaScript o passare ad altre librerie. Così scrivere:

```
var liste = $("#menu li").get();
```

equivale al codice JavaScript:

```
var liste = document.getElementById("menu").getElementsByTagName("li");
```

Se tuttavia siamo interessati solo ad un particolare elemento possiamo richiamarlo indicando a `.get()` il suo indice, in questo caso sarà restituito un riferimento diretto all'elemento del DOM:

```
var liste = $("#menu li").get(0); //solo il primo elemento della lista
```

Un metodo molto simile a questo è `.eq()`, che tuttavia **restituisce sempre un oggetto jQuery**, dandoci la possibilità di applicare su uno specifico elemento altri metodi jQuery:

```
// due modi per determinare l'html contenuto in un elemento lista

var html = $("#menu li").get(0).innerHTML; // con JavaScript nativo
var html = $("#menu li").eq(0).html(); // con metodo jQuery
```

Va precisato che dalla versione 1.3 jQuery raccoglie gli elementi **secondo l'ordine indicato nel selettore CSS** e non secondo la loro posizione nel documento:

```
<h1></h1>
<h1></h1>
```

```
<p></p>

<h2></h2>

<p></p>

$("h1,h2,p"); //ordine elementi: <h1>,<h1>,<h2>,<p>,<p>
```

Sempre legato alla posizione di un elemento è il metodo `.index()`, che ritorna **l'indice di un elemento rispetto a quelli selezionati**. Questo motodo accetta come argomento il riferimento ad un elemento del DOM:

```
<ul id="menu">
   <li id="primo">primo</li>
   <li>secondo</li>
</ul>

$("#menu li").index(document.getElementById("primo")); // ritorna 0
```

Nel caso fosse passato un oggetto jQuery verrà preso in considerazione **solo in primo elemento** collezionato, quindi l'esempio precedente potrebbe essere:

```
$("#menu li").index( $("#primo") ); // ritorna 0
```

Ciclare gli elementi

Un metodo che risulta spesso molto utile è `.each()`, che permette di applicare una funzione specifica su ogni elemento della collezione:

```
$("#menu li").each(function () {
     var id = this.id;
});
```

All'interno della funzione passata ad `.each()`, **this** rappresenta l'elemento specifico. In alternativa possiamo definire **due argomenti** per rappresentare l'elemento ed il suo indice nella collezione. Attenzione: a differenza di altre librerie il primo argomento è l'indice:

```
$("#menu li").each(function (i,el) {
        var index = i; //0,1,2 etc
  var id = el.id; // l'id dell'elemento
});
```

Come detto in precedenza (ma non è mai troppo per chi si avvicina a jQuery) per utilizzare i metodi jQuery è necessario passare da `$`, anche all'interno di funzioni come queste:

```
$("#menu li").each(function (i,el) {
  var index = i; //0,1,2 etc
  var id = $(el).attr("id"); // l'id dell'elemento con jQuery
});
```

Ecco una pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/core-method-1.html) per i metodi `.index()` e `.each()`

In realtà, come vedremo più avanti, molti dei metodi jQuery possono essere passati **direttamente** alla collezione.

Versione originale: http://javascript.html.it/guide/lezione/4150/metodi-di-base/

Metodi per gli attributi

Uno dei gruppi di metodi più potenti in jQuery è sicuramente quello relativo agli attributi. In generale il suo funzionamento prevede tre comportamenti in base agli argomenti passati:

- **1 argomento stringa**: restuituisce il valure dell'attributo
- **2 argomenti stringa**: imposta l'attributo
- **1 argomento stringa e una funzione**: imposta l'attributo in base alla funzione passata
- **1 oggetto formato da coppie `'attributo' : 'valore'`** : imposta attributi multipli

Lavorare con gli attributi

Il metodo specifico per ricavare ed impostare gli attributi degli elementi è, molto sinteticamente `.attr()`, eccone alcuni esempi introduttivi:

```
$("a#mioLink").attr("href"); //restituisce il valore di href

$("a#mioLink").attr("href","http://www.html.it"); //imposta il valore di href

$("a#mioLink").attr("href",function () { ... }); //imposta il valore di href in base alla funzione

$("a#mioLink").attr({
   "href":"http://www.html.it"
   "target":"_blank"
}); //imposta il valore di href e del target
```

Nel caso di una funzione passata come secondo argomento è possibile impostare due argomenti come con `.each()`. Due esempi abbastanza completi di come si potrebbe utilizzare questo medoto sono i seguenti:

```
//raccogliere la lista degli URL da una serie di tag a

var ListaURL = []; //array di base

$("#menu li a").each(function (i,el) {
     ListaURL.push($(el).attr("href");
});

  //impostare il link di una serie di tag a in base alla posizione

var ListaURL = ["http://www.html.it","http://www.google.it","http://www.yahoo.it"];

$("#menu li a").attr("href",function (i,el) {
     return ListaURL[i];
});
```

Nel caso volessimo invece eliminare un attributo basterà utilizzare `.removeAttr()`:

```
//rimuovere l'attributo target da tutti gli elementi della collezione
$("#menu li a").removeAttr("target");
```

Versione originale: http://javascript.html.it/guide/lezione/4151/metodi-per-gli-attributi/

Gestione delle classi, del contenuto e dei campi dei form

Gestire le classi

Uno dei problemi maggiori legati alla manipolazione degli attributi riguarda le classi ed in particolare la possibiltà di associarne più d'una ad un elemento con la conseguente necessità di poterle eliminare singolarmente senza toccare le altre. A questo problema rispondo alcuni metodi specifici jQuery:

- `.hasClass()`: metodo di controllo, ritorna true se l'elemento ha uan specifica classe
- `.addClass()`: aggiungi una classe agli elementi
- `.removeClass()`: rimuove una classe agli elementi
- `.toggleClass()`: aggiunge una classe se già non presente, altrimenti la toglie

Per tutti questi metodi è possibile indicare, oltre ad una singola classe, anche due o più classi separate da una spazio (`"miaClasse1 miaClasse2"`). Inoltre `.toggleClass()` permette anche di indicare un secondo parametro (`true` o `false`) per definire se aggiungere o togliere una classe:

```
$("#menu li").toggleClass("miaClasse",true); //aggiungi sempre la classe
```

Manipolare il contenuto

jQuery mette a disposizione due metodi specifici per manipolare il contenuto degli elementi. Anzitutto `.text()` permette di trovare o impostare il testo contenuto in un elemento, mentre `.html()` gestisce anche il codice HTML. Ecco un esempio che ne spiega le differenze:

```
<p>testo del <strong>paragrafo</strong></p>
  $("p").text(); // "testo del paragrafo"
$("p").html(); // "testo del <strong>paragrafo</strong>"
```

Ambedue i metodi restituiscono una stringa contenente il testo o l'HTML di tutti gli elementi nella collezione. Inoltre `.text()` può essere usato anche per estrarre il contenuto dai nodi di un file XML.

Come per `.attr()`, questi metodi possono anche essere utilizzati per impostare il testo o il contenuto html degli elementi nella collezione:

```
$("p").text("Nuovo testo");
$("p").html("Nuovo testo con <strong>HTML</strong>");
```

Gestione dei form

Un gruppo a sé state di metodi per gli attributi riguarda la gestione dei campi nei form. Come nei casi precedenti attraverso `.val()` sarete in grado di ottenere ed impostare i valore inseriti dall'utente, sia per quanto riguarda i campi di testo (text, password, textarea), sia per i campi select che per i campi radio e checkbox. In particolare i campi select con attributo `multiple` restituiranno un array di valori, mentre i gruppi di checkbox restituiranno solo il primo valore selezionato come si può vedere in questa pagina (http://www.html.it/guide/esempi/jquery/esempi/attributi-1.html) di esempio.

In modo analogo, per impostare i campi di un form basterà passare a `.val()` una stringa oppure un array di stringhe nel caso di valori multipli per radio checkbox e select (esempio (http://www.html.it/guide/esempi/jquery/esempi/attributi-2.html)).

Versione originale: http://javascript.html.it/guide/lezione/4152/gestione-delle-classi-del-contenuto-e-dei-campi-dei-form/

Traversing

Nelle lezioni precedenti abbiamo visto come sia possibile, attraverso i selettori CSS, ottenere una collezione di elementi da un documento HTML in modo da poterli manipolare e modificare con jQuery. In alcuni casi, tuttavia, questa collezione potrebbe risultare incompleta oppure poco *raffinata* per poterci permettere di lavorare al meglio. Inoltre potremmo pensare di utilizzarla come base per *attraversare* dinamicamente gli altri elementi del DOM e lavorarvi. A queste necessità rispondono una serie di metodi definiti *Traversing*.

Filtrare la collezione

Oltre al metodo `.eq()`(visto in precedenza), jQuery mette a disposizione alcuni metodi per filtrare la collezione di elementi. Uno dei più utili è sicuramente `.filter()` che accetta come argomento sia un selettore CSS, sia una funzione. Nel primo caso basterà passare la stringa alla funzione per eliminare tutti gli elementi che non corrispondono alle regole indicate:

```
$("#contenitore a").filter(".external"); //restituisce solo i  link con classe external
```

In alternativa si può definire una funzione che operi da filtro restituendo `true` o `false`. Adattando l'esempio precedente dovremmo scrivere:

```
$("#contenitore a").filter(function () {

        // this rappresenta un singolo elemento
  return $(this).hasClass("external");

});
```

Il metodo opposto a `.filter()` è `.not()`, che include tutti gli elementi che NON corrispondono al selettore indicato.

```
$("#contenitore a").not(".external"); //tutti i link senza classe external
```

Molto simile a `.eq()` è `.slice()`, che opera nello stesso modo del metodo per array nativo in JavaScript, estraendo un specifica porzione della collezione:

```
$("#contenitore a").slice(0, 2) // solo il primo e secondo link
```

Chiude il gruppo dei filtri `.is()`, che accetta un selettore CSS come argomento e restituisce `true` quando gli elementi rispettano le regole indicate. Questo metodo è particolamente utile nelle strutture di controllo all'interno di cicli `.each()` per suddividere le operazioni da compiere sugli elementi in base ai valori di `this`:

```
$("#contenitore a").each(function () {

  if ($(this).is(".external")) {
                //codice per link esterni
  } else {
                //codice per link interno
  }

});
```

Versione originale: http://javascript.html.it/guide/lezione/4153/traversing/

A spasso per il DOM

jQuery mette a disposizione molti metodi per spostarsi all'interno del documento HTML modificando dinamicamente il numero e la tipologia di elementi della collezione. Prima di elencarne i più importanti, è importante definire la funzione di `.end()` e `.andSelf()`.
Il primo serve a ripristinare la collezione alla sua struttura originaria, cioè a reintegrare tutti gli elementi raccolti con il selettore passato a `$()`, eccone un esempio:

```
var lista = $("#menu li"); //tutti i tag li
lista.find("a"); //tutti i tag a contenuti nei tag li
lista.end(); //ripristino i tag li
```

Questa funzione permette, in poche parole, di ritornare indietro nella selezione al punto di partenza.

Il secondo metodo da ricordare è `.andSelf()`, con il quale possiamo includere gli elementi di partenza della selezione dentro la nuova selezione generata da uno dei metodi di traversing:

```
var lista = $("#menu li"); //tutti i tag li
lista.find("a"); //tutti i tag a contenuti nei tag li
lista.andSelf(); //tutti i tag a ed i tag li
```

Andare in profondità

Nonostante vi sia un numero considerevole di metodi *traversing*, solo alcuni vengono utilizzati spesso. Anzitutto `.find()` ricerca elementi figli all'interno della collezione attuale secondo il selettore CSS passato come argomento. Diversamente `.add()` aggiunge gli elementi selezionati alla collezione corrente senza tener conto della loro posizioen nel DOM.

Molto simile a `.find()` è `.children()`, che tuttavia ricerca solo gli elementi **direttamente contenuti** (discendenti) in quelli della collezione:

```
$("#menu").find("li"); // tutti i tag li
$("#menu").children("li"); // tutti i tag li del primo livello
```

Ecco un esempio pratico (http://www.html.it/guide/esempi/jquery/esempi/traversing-1.html) di questi metodi.

Risalire l'albero DOM

Per risalire l'albero del documento sono disponibili `.parent()` e `.parents()`: ambedue ricercano gli elementi contenitori di quelli presenti nella collezione ed accettano un selettore come filtro opzionale per raffinare la ricerca. L'unica differenza è che `.parent()` si ferma al primo livello trovato, mentre `.parents()` risale tutto l'albero (esempio (http://www.html.it/guide/esempi/jquery/esempi/traversing-2.html)).

A partire dalla versione 1.3 jQuery offre inoltre il metodo `.closest()`, che opera nello stesso modo di `.parent()` includendo nella ricerca anche l'elemento corrente. Questo metodo è molto utile nel caso l'elemento originario sia `this`:

```
$(this).closest("li"); //il primo li genitore trovato anche se è this
```

Elementi contigui

Se è possibile scendere e risalire l'albero del documento è anche possibile spostarsi fra gli elementi contigui (*sibling*) che precedono o seguono quelli della collezione. Per questo scopo possiamo utilizzare rispettivamente

`.prev()` o `.next()`. Anche questi due metodi accettano un selettore CSS come filtro ed inoltre permettono di selezionare tutti gli elementi contigui con `.prevAll()` e `.nextAll()`. Infine attraverso `.siblings()` possiamo selezionare gli elementi adiacenti sia precedenti che seguenti (esempio (http://www.html.it/guide/esempi/jquery/esempi/traversing-3.html)).

Catene complesse

Una volta comprese le possibilità del traversing è anche possibile comporre catene complesse di metodi jQuery per lavorare su molti elementi partendo da una collezione di base. In questo caso è anche importante **seguire delle regole di scrittura** del codice che lo rendano leggibile. Solitamente una catena segue queste convenzioni:

```
$(selettore)
        .find(selettore).css([...])
.end()
        .find(altroselettore).attr([...])
        .next().attr([...])
```

Ricordatevi comunque che non tutti i metodi jQuery sono concatenabili, ma solo quelli che non restituiscono un valore (array stringa o boolean che sia).

```
//rimuovere l'attributo target da tutti gli elementi della collezione
$("#menu li a").removeAttr("target");
```

Versione originale: http://javascript.html.it/guide/lezione/4154/a-spasso-per-il-dom/

Inserire elementi

Ciò che rende affascinante un'applicazione web dei giorni nostri è la diffusa interazione fra gli elementi della pagina e l'utente. Poter manipolare facilmente i nodi di un documento in risposta ad un click dell'utente diventa quindi una feature fondamentale per un framework JavaScript.

Vedremo più avanti come jQuery sia in grado di gestire con efficacia le chiamate AJAX, ora invece ci focalizzeremo su come sia possibile inserire ed eliminare dinamicamente interi pezzi di HTML con poche righe di codice in piena compatibilità fra tutti i browser.

Riepilogo

Anzitutto un riepilogo: abbiamo già visto come la funzione $ accetti codice HTML per creare nuovi nodi del DOM anche con strutture nidificate. Per esempio il codice seguente creerà un nuovo elemento nell'oggetto jQuery:

```
$("<p>Nuovo <strong>paragrafo</strong></p>");
```

Allo stesso modo abbiamo visto come `.html()` e `.text()` servano per manipolare il contenuto HTML e testuale degli di una collezione:

```
$("p").html("Testo del <strong>paragrafo</strong>");
```

Questi due metodi, tuttavia, interessano tutto il contenuto di un elemento rimpiazzandolo completamente, mentre creare un nuovo elemento non significa averlo posizionato nel documento.

Nelle prossime lezioni vedremo come sia possibile inserire dinamicamente i nuovi elementi in posizioni ben precise del documento HTML e come si possano clonare ed eliminare i nodi del DOM, nonché cambiarne la posizione.

Inserire elementi annidati

Una regola che vale per tutti i metodi del gruppo *manipulation* è che **ognuno presenta la possibilità di gestire la manipolazione a partire da un contenuto preesistente oppure da un nuovo contenuto**. Ecco allora che per inserire un nuovo elemento alla fine di una lista puntata possiamo scrivere:

```
//seleziono un elemento e vi inserisco un nodo
$("#menu").append("<li>lista</li>");
```

oppure:

```
//inserisco un nuovo nodo nell'elemento selezionato
$("<li>lista</li>").appendTo("#menu");
```

Questi stessi metodi possono risultare utili per spostare un elemento da una posizione all'altra nel DOM:

```
//sposta i nodi da una lista all'altra
$("#menu2 li").appendTo("#menu");
```

Analogamente, è possibile anche inserire nuovi nodi o spostarli all'inizio di un elemento utilizzando `.prepend()` e `.prependTo()`

Ecco una pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/07-manipulation-1.html).

Versione originale: http://javascript.html.it/guide/lezione/4249/inserire-elementi/

Inserire elementi contigui e avvolgere elementi

Inserire elementi contigui

Con questo gruppo di metodi possiamo inserire elementi prima o dopo un determinato punto del DOM nello stesso modo del gruppo precedente. I metodi da utilizzare sono `.after()` / `.before()` e `.insertAfter()` / `.insertBefore()`:

```
//due modi per inserire una nuova lista DOPO una già presente
$("#menu").after("<ul id='menu2'></ul>");
$("<ul id='menu2'></ul>").insertAfter("#menu");

//due modi per inserire una nuova lista PRIMA di una già presente
$("#menu").before("<ul id='menu2'></ul>");
$("<ul id='menu2'></ul>").insertBefore("#menu");
```

Avvolgere gli elementi

Attraverso questi metodi è possibile *avvolgere* gli elementi della collezione con una struttura HTML predefinita (sia nuova che preesistente nel documento). Il metodo principale di questo gruppo è `.wrap()` che accetta come argomento sia codice HTML sia un riferimento ad un elemento specifico:

```
$("p").wrap("<div></div>");
$("p").wrap(document.createElement("div"));
```

Con il codice precedente avvolgeremo **ogni paragrafo** in un tag `div`. Se invece vogliamo raccogliere gli elementi della collezione all'interno di un contenitore dobbiamo usare `.wrapAll()`:

```
$("p").wrapAll("<div></div>");
$("p").wrapAll(document.createElement("div"));
```

Con quest'ultimo metodo gli elementi della collezione vengono raggruppati in corrispondenza del primo elemento trovato e quindi avvolti. Ecco un esempio (http://www.html.it/guide/esempi/jquery/esempi/07-manipulation-2.html).

Un ultimo metodo che lavora sui nodi figli degli elementi è `.wrapInner()`, molto utile nel caso volessimo nidificare parti del documento senza dover agire direttamente sull'HTML di origine (magari per applicare particolari stili CSS):

```
<p>Testo</p>

$("p").wrapInner("<span></span>");

<p><span>Testo</span></p>
```

Come gi altri metodi del gruppo, anche `.wrapInner()` accetta riferimenti diretti ad elementi.

Versione originale: http://javascript.html.it/guide/lezione/4250/inserire-elementi-contigui-e-avvolgere-elementi/

Sostituire, rimuovere, svuotare e duplicare

In alcuni casi, più che aggiungere o spostare gli elementi potrebbe essere necessario rimpiazzarne alcuni con altri, sia nuovi che già presenti nella pagina. A questo scopo il metodo da utilizzare è `.replaceWith()`, che accetta come argomento un oggetto jQuery oppure semplice codice HTML (esempio (http://www.html.it/guide/esempi/jquery/esempi/07-manipulation-3.html))

```
//rimpiazza l'elemento segnaposto
$("#segnaposto").replaceWith("<p>Nuovo contenuto</p>");
$("#segnaposto").replaceWith($("p#nuovoContenuto"));
```

Da notare che **l'oggetto jQuery restituito conterrà comunque nella sua collezione l'elemento appena sostituito**:

```
$("div:first").replaceWith("<p></p>").is("div");
//restituisce true!
```

Un'altro metodo per la sostituzione è `.replaceAll()`, che inverte l'ordine degli argomenti percui tutti gli elementi della collezione sostutuiranno quelli definiti dal selettore CSS passato al metodo:

```
//rimpiazza l'elemento segnaposto
$("<p>Nuovo contenuto</p>").replaceAll("#segnaposto");
```

Rimuovere, Svuotare e Duplicare

Gli ultimi metodi del gruppo assolvono ad alcune funzioni fondamentali per una moderna applicazione web rendendo come sempre il processo molto semplice.
Anzittutto com `.empty()` possiamo svuotare un elemento di tutto il suo contenuto; utilizzeremo questo metodo, ad esempio, per cancellare il testo di un box a comparsa:

```
$("#messaggio").empty();
```

Nel caso invece volessimo rimuovere completamente un elemento utilizzeremo `.remove()`, passandogli eventualmente un selettore per filtrare gli elementi da eliminare:

```
$("table #riga1").remove();
//equivale a
$("table tr").remove("#riga1");
```

Sebbene i due esempi qui sopra possano sembrare uguali è bene ricordare che nel primo caso jQuery **manterrà un riferimento** nella collezione all'elemento `#riga1`, sul quale potremo comunque continuare a lavorare anche se non esiste più*fisicamente* nel documento:

```
$("table #riga1").remove().attr("id");
//restituisce "riga1"
```

Chiude il gruppo il metodo `.clone()`, che molto semplicemente duplica gli elementi selezionati e li ridefinisce come elementi della collezione. Poiché questo metodo si limita a clonare, dovremo utilizzare gli altri metodi del gruppo manipulation per inserire i cloni all'interno del documento:

```
//clonare una lista
$("#menu li").clone().appendTo("#menu2");
```

Questo metodo, inoltre, **non copia gli eventi associati agli elementi** della collezione se non viene indicato

esplicitamente passandogli `true` come argomento:

```
//clonare una lista con gli eventi JavaScript associati
$("#menu li").clone(true).appendTo("#menu2");
```

Ecco alcuni esempi dimostrativi (http://www.html.it/guide/esempi/jquery/esempi/07-manipulation-4.html)

Versione originale: http://javascript.html.it/guide/lezione/4262/sostituire-rimuovere-svuotare-e-duplicare/

Manipolare i CSS

Una delle funzioni più amate ed utilizzate dei framework JavaScript è senza dubbio la manipolazione degli stili CSS e della posizione degli elementi. Questo perché JavaScript nativo risente fortemente delle differenze nella sintassi e nelle reazioni dei vari browser.

Manipolare gli Stili

In jQuery il metodo principale è `.css()` che, analogamente ad `.attr()`, permette di ottenere o modificare gli stili CSS di un elemento:

```
$("a").css("color"); //restituisce il colore esadecimale del primo elemento link

$("a").css("color","#FF0000"); //imposta il colore dei link

$("a").css({
        "color" : "#FF0000", //imposta il colore
        "display" : "block" // imposta la visualizzazione
});
```

Come si vede negli esempi, il metodo lavora con gli stessi nomi delle regole CSS dei fogli di stile. In realtà accetta anche i nomi che usa JavaScript per indicare alcune regole, come quelle degli sfondi:

```
//metodi equivalenti per impostare il colore di sfondo
$("a").css("background-color","#FF0000");
$("a").css("backgroundColor","#FF0000");
```

Posizione e dimensione

Oltre a `.css()`, il framework dispone di alcuni metodi specifici relativi alle dimensioni e alla posizione degli elementi, che permettono di impostare o ottenere alcuni stili con facilità:

- `.width()` - `.height()`: permette di trovare o impostare larghezza e altezza complessiva di un elemento in pixel
- `.innerWidth()` - `.innerHeight()`: larghezza e altezza interne di un elemento (include il padding, non conta bordi e margini)
- `.outerWidth()` - `.outerHeight()`: larghezza e altezza esterne di un elemento (conta bordi e padding, opzionalmente i margini passando l'argomento `true`)

- `.offset()`: ritorna un oggetto con la distanza da sinistra e dall'alto dell'emento rispetto al documento
- `.position()`: come `.offset()` ma le distanze sono relative al contenitore più prossimo
- `.scrollTop()` - `.scrollLeft()`: trova o imposta scroll dell'elemento rispetto al documento

A parte `.offset()` e `.position()`, tutti questi metodi si possono applicare anche ad elementi nascosti, in modo da poter ottenere ed impostare posione e dimensione prima di mostrarli. Ecco alcuni esempi pratici (http://www.html.it/guide/esempi/jquery/esempi/css-1.html).

Versione originale: http://javascript.html.it/guide/lezione/4263/manipolare-i-css/

Gestire gli eventi

La gestione degli eventi in JavaScript rappresenta un altro aspetto tanto essenziale per l'interazione utente quanto mal digerito dai browser. Proprio per questo jQuery fornisce strumenti cross-browser particolarmente avanzati per gestire gli eventi e la loro relazione con gli elementi del DOM. Una spiegazione dettagliata potrebbe risultare lunga e forse poco pratica, considerando che in generale il tipo di interazione più diffuso rimane sempre il classico click del mouse. Per iniziare vediamo un esempio:

```
$("a").bind("click",function (event) {
                alert($(this).attr("href"));
});
```

Questo esempio associa ad ogni tag a un evento che farà visualizzare il valore `href` del link specifico.

Gestire gli eventi

Un comportamento non molto diverso dall'esempio precedente l'avremmo ottenuto scrivendo la funzione dentro l'attributo `onclick` del tag. Quello che jQuery da in più è la capacità di gestire in modo semplice l'evento stesso. Anzitutto è da notare che `this` rappresenta sempre l'elemento a cui è associato l'evento e che non sempre è quello su cui si è realmente cliccato; questo avviene perché jQuery (rispettando la gestione eventi nativa in JavaScript) tenta di risalire l'albero del documento, attivando l'evento specifico nel primo elemento trovato. Un esempio:

```
<a href="index.html"><strong>Testo</strong></a>

$("a").bind("click",function (event) {
     alert($(this).attr("href"));
});
```

Quando clicchiamo sul link sopra, in realtà l'elemento su cui agiamo è il tag `strong` al quale tuttavia non è associato alcun evento. Quindi jQuery risale l'albero del DOM, trova l'evento che abbiamo associato al tag a e lo lancia. Questo è un modo intelligente per gestire gli elementi annidati slegandosi dal focus reale del nostro mouse.
Nel caso volessimo sapere qual'è l'elemento realmente cliccato potremo basarci sulla proprietà `.target` dell'oggetto `event` che abbiamo passato come argomento alla funzione precedente:

```
$("a").bind("click",function (event) {
                var target = event.target;
                alert(target.tagName); //il nome del tag su cui abbiamo cliccato
});
```

Un'altra proprietà interessante dell'oggetto `event` è `.type`, che indica sotto forma di stringa il tipo di evento che è stato lanciato. Utilizzando questa proprietà potremmo per esempio creare una funziona generica in cui, con un costrutto `switch` o `if else`, eseguire del codice in base ad eventi specifici:

```
function lanciaEvento (event) {
     if (event.type == "click") {
                alert ("Click!");
     } else {
                alert ("Un altro evento");
     }
}

$("a").bind("click focus blur",function (event) {
     lanciaEvento(event);
});
```

Oltre a darci la possibilità di trovare importanti informazioni sugli eventi, l'oggetto `event` ci permette di agire direttamente sul loro comportamento, per esempio bloccandone l'azione predefinita. Per fare un esempio: cliccando su un link l'azione predefinita del browser sarà quella di caricare la pagina indicata nell'attributo `href`. Per bloccare quest'azione predefinita, spesso si ricorre ad uno script del genere:

```
<a href="link.html" onclick="return false;">clicca qui</a>
```

Con jQuery diventa invece indolore utilizzare una funzione specifica di JavaScript ma non implementata in IE `.preventDefault()`:

```
$("a").bind("click",function (event) {
                event.preventDefault(); //blocca l'evento di default
                // codice da eseguire
});
```

La stessa cosa può risultare utile per inviare i dati di un form con AJAX modificando il comportamento di un pulsante submit pur mantenendo la possibilità di inviare i dati normalmente nel caso JavaScript sia disattivato:

```
$(":submit").bind("click",function (event) {
                event.preventDefault(); //blocca l'evento di default
                var action = $(this).parents("form").attr("action"); //determina il valore di action
                //codice AJAX qui
});
```

Versione originale: http://javascript.html.it/guide/lezione/4270/gestire-gli-eventi/

Associare gli eventi: bind e unbind

Negli esempi precedenti abbiamo usato `.bind()` per associare delle funzioni (dette *handlers*) agli eventi. Questo metodo prevede come primo argomento una stringa che definisca l'evento a cui associare una funzione oppure più eventi separati da uno spazio. Gli eventi supportati sono: `blur`, `focus`, `load`, `resize`, `scroll`, `unload`, `beforeunload`, `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `mouseenter`, `mouseleave`, `change`, `select`, `submit`, `keydown`, `keypress`, `keyup`, `error`.
Come secondo agomento `.bind()` accetta sia una funzione (come già visto in precedenza) oppure un oggetto JavaScript con dei dati ricavabili dalla proprietà `.data` di `event`. In quest'ultimo caso la funzione da associare diventa il terzo agomento:

```
$(":text").bind("focus blur", {"background-color" : "#FFFFCC"}, function (event) {
        if (event.type == "focus") {
                $(this).css(event.data);
        } else {
                $(this).css("background-color","");
        }
});
```

Ecco questo script in azione (http://www.html.it/guide/esempi/jquery/esempi/09-eventi-1.html).

Lanciare e Cancellare le associazioni

Come è possibile associare degli *handlers*, allo stesso modo è possibile rimuoverli oppure lanciarli senza che l'evento sia realmente accaduto. Nel primo caso utilizzeremo `.unbind()`, nel quale possiamo specificare sia gli eventi da prendere in considerazione, sia specifiche funzioni da rimuovere:

```
$(":text").unbind(); // rimuovi tutti gli eventi dai campi di testo
$(":text").unbind("blur focus"); //rimuovi gli handlers per gli eventi focus e blur
$(":text").unbind("blur",blurFunction); // rimuovi uno specifico handler per l'evento blur
```

Ecco l'esempio precedente con la possibilità di eleminare le associazioni (http://www.html.it/guide/esempi/jquery/esempi/09-eventi-2.html).

Nel caso in cui volessimo lanciare un evento specifico, invece, jQuery mette a disposizione due metodi: `.trigger()` e `.triggerHandler()`. Nella pratica questi metodi sono utili per aprire una galleria immagini da un link diverso da una miniatura:

```
$("#lanciaGalleria").bind("click",function (event) {
                event.preventDefault();
                $('a.galleria:first').trigger('click');
});
```

Questi due metodi producono lo stesso effetto, tuttavia la differenza sta nel fatto che `.triggerHandler()` **non lancia gli eventi predefiniti dell'elemento e agisce solo sul primo elemento di una collezione**.
L'esempio precedente può allora essere riscritto così:

```
$("#lanciaGalleria").bind("click",function () {
                $('a.galleria').triggerHandler('click');
});
```

Versione originale: http://javascript.html.it/guide/lezione/4271/associare-gli-eventi-bind-e-unbind/

Metodi speciali e 'event delegation'

Oltre a `.bind()` jQuery offre alcuni metodi specifici:

- `.one()`: funziona come `.bind()` ma esegue l'*handler* associato all'evento **solo una volta**
- `.hover()`: accetta due funzioni, una per quando il mouse è sopra un elemento ed una per quando lo lascia
- `.toggle()`: accetta una o più funzioni come argomento, lanciandole in sequenza **ad ogni click**

Ecco gli esempi (http://www.html.it/guide/esempi/jquery/esempi/09-eventi-3.html).

Una nota particolare riguarda `.ready()`, che associato a `document` lancia una funzione quando l'abero del DOM è stato caricato, senza attendere il caricamento di immagini e fogli di stile:

```
$(document).ready(function () {
                //codice
});
```

Come già visto nelle prime lezioni, questo metodo può anche essere riscritto come:

```
$(function () {
                //codice
});
```

Alcune scorciatoie

E' quasi inutile ripetere ancora una volta che jQuery ha a cuore la sinteticità del codice. Proprio per questo anche per gli eventi sono disponibili delle scorciatoie come:

```
$("a").click(function () {
                //codice da associare
});
```

```
$("a").click(); //lancia gli handler associati ad a per l'evento click
```

Per una lista completa delle scorciatoie vi rimando alla documentazione ufficiale.

Event Delegation

Una delle novità introdotte con la versione 1.3 è il supporto nativo per l'*event delegation*. Con questo termine ci si riferisce alla possibilità di associare un *handler* ad un evento anche per elementi non ancora presenti nel documento. Infatti, quando utilizziamo `.bind()` su una collezione di elementi, l'*handler* non viene associato ai nuovi elementi aggiunti dinamicamente, ma solo a quelli già esistenti.
Il metodo specifico per l'event delegation è `.live()`, con il corrispettivi `.die()` per cancellare l'associazione. Al momento sono presenti alcune limitazioni:

- è possibile associare solo un evento alla volta e **solo un sottogruppo degli eventi** disponibili con `.bind()`
- `.live()` **si applica solo direttamente ad un selettore** (non è possibile associarlo dopo aver filtrato o attraversato la collezione)

Un esempio tipico dell'utilizzo di questo metodo è:

```
$("#lista li").live("click",function () {
```

```
                //codice da eseguire
});
```

Versione originale: http://javascript.html.it/guide/lezione/4272/metodi-speciali-e-event-delegation/

La coda degli effetti

La "rinascita" di JavaScript e la sua diffusione nel web 2.0 è senza dubbio legata alle librerie di animazione. Passando per Scriptaculous e Moo.Fx, la possibilità di creare effetti dinamici senza ricorrere ad interfacce interamente realizzate in Flash ha rappresentato per molti sviluppatori una valida ragione per affacciarsi a JavaScript.

Anche jQuery incorpora una serie di metodi nativi per la realizzazione di effetti grafici.

La coda degli effetti.

Una caratteristica comune a tutti gli effetti di jQuery è quella di essere **inseriti di default in una coda generale**. Questo approccio garantisce che ogni animazione parta quando la precedente è conclusa, in modo da evitare sovrapposizioni, errori o brutti sfarfallii sullo schermo.

Nonostante la coda degli effetti sia predefinita, è possibile interagirvi attraverso due metodi specifici: `.queue()` e `.dequeue()`.

Passando a `.queue()` una stringa con il nome della coda (`"fx"` di default) è possibile anzitutto determinare quali funzioni siano presenti nella coda:

```
var coda = $("div").queue("fx");
//Il valore restituito è un array di funzioni
```

Passando invece una funzione, questa verrà eseguita alla fine della coda. Ecco un codice di esempio:

```
function effetti () {
    //effetto 1
    $("div").show("slow");

    //effetto 2
    $("div").animate({left:'+=200'},2000);

    //alla fine dell'effetto 2 esegui questa funzione
    $("div").queue("fx",function () {
        $(this).addClass("nuovaClasse");
        //.dequeue() riavvia la coda
        $(this).dequeue();
    });

    //effetto 3
    $("div").animate({left:'-=200'},500);
    //effetto 4
    $("div").slideUp();
}
```

Una volta che la funzione `effetti()` sarà lanciata, la coda di default `"fx"` verrà riempita man mano con le animazioni e verrà interrotta a metà da `.queue()` per poi essere ripresa grazie a `.dequeue()`.

Nel caso volessimo invece rimpiazzare la coda di effetti, possiamo indicarli all'interno di un array e passarlo al

metodo:

```
//Nuova coda da sostituire con quella corrente
var nuovaCoda = [
        function () { $(this).show() }, //mostra elemento
        function () {$(this).hide() } //nascondi elemento
];

//Sostituisco la coda
$("div").queue("fx",nuovaCoda);

//Cancello la coda con un array vuoto
$("div").queue("fx",[]);
```

Come visto nell'esempio precedente il metodo `.dequeue()` viene utilizzato per lanciare immediatamente la prossima funzione nella coda. Anche questa funzione accetta come parametro il nome della coda, che è di default sempre `"fx"`.

Versione originale: http://javascript.html.it/guide/lezione/4391/la-coda-degli-effetti/

Effetti di base

Passiamo ora ai metodi disponibili per generare animazioni. Il primo gruppo comprende alcuni effetti di base, richiamabili attraverso una sintassi sintetica ed intuitiva, ecco un esempio:

```
//mostra tutti gli elementi p nascosti
$("p").show("slow");
```

Con il metodo `.show()` tutti gli elementi della collezione vengono mostrati **animandone contemporaneamente l'opacità e la dimensione**. L'operazione opposta viene eseguita da `.hide()`, mentre per alternare le due animazioni in base alla visibilità degli elementi si può usare `.toggle()`.

Per quanto riguarda gli argomenti da passare, tutti i metodi accettano anzitutto una stringa rappresentante la **durata generica dell'animazione** (`slow`, `normal`, `fast`) oppure uno specifico **periodo in millisecondi**. Inoltre si può associare una **funzione di callback** da eseguire alla fine dell'animazione. Volendo introdurre un esempio completo del metodo `.show()` visto in precedenza potremmo scrivere:

```
$(".nascosto").show("slow",function () {
        alert("Animazione conclusa!");
});
```

Una caratteristica interessante di questi tre metodi è che se non viene passato alcun argomento, **impostano direttamente lo stato finale dell'animazione,** risultando molto utili come scorciatoie per mostrare o nascondere gli elementi di una collezione:

```
$("p").hide();
// equivale a scrivere
$("p").css("display","none");
```

Ecco alcuni esempi pratici (http://www.html.it/guide/esempi/jquery/esempi/10-animazioni-1.html).

Metodi specifici

Oltre ai metodi come `.show()` che utilizzano un'animazione combinata di opacità e dimensione, jQuery mette a disposizione altre due scorciatoie per effetti molto usati: anzitutto `.slideDown()` `.slideUp()` e `.slideToggle()`, che agiscono sull'**altezza degli elementi**, quindi `.fadeIn()`, `.fadeOut()` e `.fadeTo()`, che ne modificano l'**opacità**.

Tutti questi metodi accettano i medesimi argomenti degli effetti di base, ad eccezione di `.fadeTo()` che prevede un argomento aggiuntivo indicante l'opacità finale dell'elemento:

```
//nascondi gli elementi p
$("p").fadeOut("slow");

//dimezza l'opacità di un elemento
$("p").fadeTo("slow",0.5,function () {
        alert("Animazione conclusa!");
});
```

Ecco una pagina di esempio (http://www.html.it/guide/esempi/jquery/esempi/10-animazioni-2.html).

Versione originale: http://javascript.html.it/guide/lezione/4392/effetti-di-base/

Animazioni personalizzate

L'ultimo metodo relativo alle animazioni è `.animate()`, con il quale è possibile realizzare effetti personalizzati. Nella sua forma più semplice accetta come primo argomento un oggetto JavaScript con la **proprietà CSS da animare ed il suo valore finale** (assoluto o relativo):

```
$("p").animate({
        "borderWidth" : "4px", //bordo a 4 pixel
        "width" : "+20px" //aumenta la larghezza di 200 pixel
});
```

Come è possibile vedere nell'esempio le proprietà CSS costituite da due parole (come `border-width`) vanno indicate nella loro notazione JavaScript (detta anche *camel case*)

Gli altri argomenti accettati sono la **velocità**, l'**andamento** (*easing*) ed una **funzione di callback**. Per quanto riguarda l'*easing*, il framework offre due opzioni predefinite, `linear` e `swing`, mentre altre possibili valori necessitano l'utilizzo di un plugin (http://gsgd.co.uk/sandbox/jquery/easing/):

```
$("#box").animate({
        "borderWidth" : "4px", //bordo a 4 pixel
        "width" : "+=20px" //aumenta la larghezza di 20 pixel
        },
        "slow", //animazione lenta
        "linear", //andamento lineare
        function () {
                //funzione di callback
                alert("Animazione conclusa!");
        }
);
```

Ecco questo esempio in azione (http://www.html.it/guide/esempi/jquery/esempi/10-animazioni-3.html).

In alternativa è possibili passare ad `.animate()` un secondo argomento nella forma di oggetto JavaScript contentente parametri aggiuntivi oltre a quelli accettati nella sintassi precedente. Ecco la lista completa:

- `duration`: la durata dell'animazione in millisecondi o come stringa (`slow`, `normal`, `fast`)
- `easing`: l'andamento, `linear` o `swing` (default: `swing`)
- `complete`: una funzione da lanciare quando l'animazione è conclusa
- queue: `true` o `false`, inserisce l'animazione nella coda generale degli effetti (default: `true`) oppure la esegue direttamente

Fermare e disabilitare gli effetti

Per concludere, tutti gli effetti possono essere sia **disabilitati globalmente** impostando su `true` la proprietà `jQuery.fx.off`, oppure bloccati "localmente" attraverso il metodo `.stop()`. Senza passare alcun argomento, questa funzione blocca l'effetto corrente e passa al successivo, mentre per bloccare definitivamente le animazioni (cancellandone la coda) basta passare come primo parametro `true`:

```
$("div").stop(); //ferma l'effetto corrente
$("div").stop(true); //ferma e cancella la coda
```

Infine, passando un secondo argomento `true`, l'animazione salterà direttamente allo stato finale, lanciando

anche la funzione di callback.

```
$("div").stop(true,true); //ferma e cancella la coda
```

Versione originale: http://javascript.html.it/guide/lezione/4393/animazioni-personalizzate/

Eventi locali e globali

Al giorno d'oggi, poter contare su tecniche come AJAX permette di realizzare interazioni con l'utente paragonabili a quelle di un software desktop per reattività e tempi di caricamento. Tuttavia la possibilità di caricare informazioni in modo asincrono si scontra spesso con le differenti implementazioni dei browser e i differenti tipi di dati caricati (XML, JSON, HTML).

Tutte le moderne librerie JavaScript hanno risposto a questo problema con delle implementazioni cross-browser capaci di gestire in modo semplice tutti i vari aspetti di AJAX.

jQuery implementa AJAX attraverso una serie di metodi statici dell'oggetto $. Nonché con una serie di metodi per gestire eventi AJAX sugli elementi della collezione.

Eventi AJAX

Per comprendere il modo in cui jQuery gestisce AJAX bisogna introdurre gli **eventi AJAX**. Questi sono una serie di eventi che si susseguono prima, durante e dopo una richiesta AJAX e che vengono suddivisi in due categorie:

- **locali**: eventi che si verificano all'interno di una chiamata e che possono essere impostati solo all'interno di una funzione di $.
- **globali**: eventi che coinvolgono tutto il DOM e che quindi possono essere rintracciati ed impostati su una collezione di elementi con metodi come `.bind()`.

Riporto di seguito la lista degli eventi AJAX gestiti da jQuery nell'ordine in cui avvengono durante una chiamata (tra parentesi il tipo di evento):

- `ajaxStart` (globale): quando la chiamata viene inizializzata
- `beforeSend` (locale) e `ajaxSend` (globale) : prima di inviare la chiamata
- `success` (locale) e `ajaxSuccess` (globale): quando una chiamata ha successo
- `error` (locale) e `ajaxError` (globale): quando avviene un errore
- `complete` (locale) e `ajaxComplete` (globale): quando la chiamata si conclude **indipendentemente da errori**
- `ajaxStop` (globale): quando non ci sono più chiamate da effettuare

Questa lista di eventi ci servirà per capire quali funzioni associare alle chiamate AJAX per personalizzare il modo in cui viene contattato il server e come manipolare i dati restituiti.

Inoltre, come detto in precedenza, gli eventi globali vengono lanciati **su tutti gli elementi del DOM** e per questo possono essere impostati attraverso il metodo `.bind()` o con specifiche scorciatoie:

```
$("#stato").bind("ajaxComplete",function () {
    //codice da eseguire
});
$("#stato").ajaxComplete(function () {
    //codice da eseguire
});
```

Poiché gli eventi globali risalgono il DOM a partire dall'elemento direttamente coinvolto nella richiesta, possono risultare utili per lanciare una **funzione generica per ogni chiamata Ajax**, come un avviso stile "messaggio di caricamento". Basta applicare la funzione all'elemento `document`:

```
$(document)
```

```
    .ajaxStart(
        function () {
            $("#messaggio").text("caricamento...");
        }
    )
    .ajaxStop(
        function () {
            $("#messaggio").text("");
        }
);
```

Versione originale: http://javascript.html.it/guide/lezione/4412/eventi-locali-e-globali/

jQuery Ajax

La funzione principale per inviare richieste AJAX è il metodo statico `$.ajax()`. Dati i molti aspetti della chiamata che possono essere personalizzati, la funzione accetta un'unico oggetto JavaScript con i parametri di base ed altri necessari per sovrascrivere i valori di default.

Parametri di base

- `url`: l'indirizzo al quale inviare la chiamata
- `success`: funzione da lanciare se la richiesta ha successo. Accetta come argomenti i dati restituiti dal server (interpretati di default come html o xml) e lo stato della chiamata
- `error`: funzione lanciata in caso di errore. Accetta un riferimento alla chiamata `XMLHttpRequest`, il suo stato ed eventuali errori notificati

Con questi tre parametri è possibile impostare una prima semplice chiamata AJAX di esempio:

```
$.ajax({
    url : "mioserver.html",
    success : function (data,stato) {
        $("#risultati").html(data);
        $("#statoChiamata").text(stato);
    },
    error : function (richiesta,stato,errori) {
        alert("E' evvenuto un errore. Il stato della chiamata: "+stato);
    }
});
```

Nell'esempio precedente, se la chiamata ha successo i dati verranno inseriti all'interno di specifici elementi DOM, altrimenti verrà mostrato un messaggio di errore.

Parametri opzionali e valori predefiniti

Oltre alla possibilità di impostare come opzione di `$.ajax()` tutti gli eventi locali visti in precedenza, è anche possibile personalizzare ulteriormente la chiamata AJAX relativamente ai dati inviati, la modalità di invio e il tipo di valori restituiti (tra parentesi i valori predefiniti):

- async (true): definisce se la chiamata deve essere asincrona o sincrona (nel cui caso bloccherà la pagina fino alla fine della chiamata)
- cache (true): se impostato su `false` forza la il browser a ricaricare ogni volta i dati del server anche se non sono cambiati
- `contentType` ("application/x-www-form-urlencoded"): il tipo di contenuto inviato al server
- `data`: un oggetto `{chiave : valore, chiave2 : valore}` oppure una stringa `"chiave=valore& chiave2=valore2"` contenente dei dati da inviare al server
- `dataType` ("html" o "xml"): il tipo di dati restituiti dal server. Oltre a `html`, `xml` e `text`, accetta `script` (script JavaScript), `json` e `jsonp` (oggetti JavaScript da server locali e remoti)
- `global` (true): se impostato su `false` gli eventi AJAX globali non verranno lanciati durante questa richiesta
- `ifModified` (false): permette che la richiesta sia considerata conclusa con successo (eventi `success` e `ajaxSuccess`) solo se i dati restituiti sono diversi da quelli ricavati da una precedente chiamata dello stesso tipo

- `jsonp`: sovrascrive il valore callback in una strigna di richesta jsonp. Così con `{jsonp:'onJsonPLoad'}` verrà aggiunta alla stringa passata al server `"onJsonPLoad=?"`
- `username`: nome utente se richiesto dal server
- `password`: password se richiesta dal server
- `processData` (true): se impostato su false i dati inviati al server non saranno pre-processati ma inviati direttamente
- `scriptCharset`: forza il charset utilizzato nella richiesta AJAX per i dataType `script` e `jsonp`
- `timeout`: un numero indicante i millisecondi dopo i quali la richiesta viene considerata scaduta
- `type` ("GET"): il tipo di richiesta da effettuare. Accetta `GET` o `POST`
- `xhr`: una funzione per personalizzare l'oggetto `XMLHttpRequest` nel caso non si volesse utilizzare ActiveXObject per Internet Explorer o XMLHttpRequest per gli altri browser

Conoscendo questi parametri sarà quandi possibile richiedere dei dati in formato JSON da inserire in una tabella HTML (esempio):

```
$.ajax({
    url : 'dati.php',
    data : 'primariga=0&ultimariga=10', //le prime 10 righe
    dataType : 'json', //restituisce un oggetto JSON
    complete: function (righe,stato) {
        for (i=0; i < righe.length; i++) {
                        var riga = righe[i];
                                $("<tr/>")
                        .append("<td>"+riga.colonna1+"</td><td>"+riga.colonna2+"</td>")
                        .appendTo("#tabella");
                }

    }
});
```

Ecco questo script in azione (http://www.html.it/guide/esempi/jquery/esempi/11-ajax-1.html).

Versione originale: http://javascript.html.it/guide/lezione/4413/jquery-ajax/

Scorciatoie e funzioni di supporto

Oltre a `$.ajax()`, jQuery mette a disposizione alcune utili scorciatoie per alcuni tipi di richieste molto utilizzate.

Anzitutto `$.post()` e `$.get()` che inviano rispettivamente richieste POST e GET al server. Ambedue le funzioni accettano fino a 4 parametri:

- `url`: l'url a cui inviare la chiamata
- `data`: come in `$.ajax()`, un oggetto o stringa rappresentante i valori da inviare al server
- `callback`: una funzione da lanciare quando i dati sono stati caricati correttamente. Accetta come argomento i dati e lo stato della richiesta
- `type`: il tipo di dati richiesto, di default html o xml (vedi `dataType` in `$.ajax()`).

Altre due scorciatoie molto utili sono `$.getJSON()` e `$.getScript()`. La prima serve a **caricare dati JSON** da un server (anche remoto) ed accetta come argomenti l'indirizzo della richiesta, eventuali dati in formato stringa e JSON, ed una funzione di callback (dello stesso tipo di `$.get()` e `$.post()`); la secondo serve a **caricare JavaScript remoto** ed accetta solo l'indirizzo del server ed una funzione di callback. Un'ultima funzione è `.load()`, un metodo che gestisce solo dati HTML o testuali e li inserisce all'interno degli elementi della collezione corrente. Come altre scorciatoie accetta anche dei dati da inviare al server come stringa o oggetto JSON, nonché una funzione di callback da lanciare al caricamento dei dati:

```
$("#pannello").load(
    "pagina.html", //pagina da caricare
    {}, //un oggetto JavaScript vuoto = nessun dato da inviare
    function () { //funzione di callback
        alert("dati caricati!");
    }
});
```

Ecco un esempio pratico (http://www.html.it/guide/esempi/jquery/esempi/11-ajax-2.html).

Funzioni di supporto

Come detto sopra, `$.ajax()` accetta come argomento un oggetto con alcuni parametri per definire il comportamento della richiesta AJAX.
Nel caso questi parametri dovessero essere impostati per molte richieste, è possibile **definirli globalmente** passandoli, sempre come oggetti JavaScript, alla funzione `$.ajaxSetup()`:

```
//Tutte le richieste AJAX restituiscono un oggetto JSON
$.ajaxSetup({
    dataType : 'json'
});
```

Per quanto riguarda la gestione dei dati da inviare via AJAX, jQuery offre due interessanti metodi: il primo, `.serialize()`, converte i campi di un form in una stringa serializzata:

```
var stringa = $("form").serialize();
//stringa = "nomecampo1=valore&nomecampo2=valore"
```

Il secondo metodo è `.serializeArray()` che si applica sempre ad un form, ma restituisce un oggetto JSON manipolabile:

```
var oggetto = $("form").serializeArray();
//stringa = [{ key : "nomecampo1" , value : "valore1"},{ key : "nomecampo2" , value : "valore2"}]
```

Versione originale: http://javascript.html.it/guide/lezione/4414/scorciatoie-e-funzioni-di-supporto/

Lavorare con gli array

brerie, jQuery non offre un gruppo troppo nutrito di funzioni di *utilità generica*. Questo approccio segue l'idea secondo cui non serve reinventare ciò che è già disponibile. La conseguenza diretta di tutto ciò è un minor peso della libreria ed un maggior affidamento ai metodi JavaScript nativi.

Lavorare sugli Array

Il primo gruppo di utility riguarda gli array, con particolare riguardo alla possibilità di ciclare fra i valori.

La funzione `$.map()` cicla fra gli elementi di un array passato come primo argomento applicando su ognuno la funzione passata come secondo argomento. Questa funzione, a differenza di altri metodi jQuery, accetta **un primo argomento indicante il valore dell'elemento ed una seconda con il suo indice nell'array**:

```
$.map(["uno","due","tre"],function (indice,valore) {
    return indice+valore;
});
//restituisce un array ["0uno","1due","2tre"];
```

Molto simile a `$.map()` è `$.grep()` che tuttavia ha la specifica funzione di filtrare gli elementi in base al valore restituito da una funzione specifica. Inoltre in `$.grep()` la posizione degli argomenti della funzione di filtraggio è invertita:

```
var lettere = ["a","aa","aaa"];
$.grep(lettere, function (valore,indice) {
        return this.length >= 2;
});
//lettere è ["aa","aaa"]
```

Se viene passato un terzo argomento `false`, verranno restituiti i valori che non verificano la condizione:

```
var lettere = ["a","aa","aaa"];
$.grep(lettere, function (valore,indice) {
        return this.length >= 2;
},false);
//lettere è ["a"]
```

Se quello che ci interessa è invece sapere se un valore è presente all'interno di un array useremo `$.inArray()` passandogli il valore da ricercare e l'array in cui cercarlo:

```
$.inArray(4,[2,3,4]);
//restituisce true
```

Per unire due array, infine, è disponibile `$.merge()`:

```
$.merge( [0,1,2], [2,3,4] )
//restituisce [0,1,2,2,3,4]
//i duplicati non sono eliminati!
```

La maggior parte delle funzioni presentate finora opera solo con array, tuttavia potrebbe presentarsi il caso in cui non si sia sicuri del tipo di variabile che possediamo o comunque si voglia convertirla in array. In tal caso useremo `$.makeArray()`:

```
//un array rimane tale
```

```
$.makeArray([0,1,2]);
// [0,1,2]

//un oggetto diventa l'unico valore di un array
$.makeArray({"chiave1":"valore1"});
// [{"chiave1":"valore1"}]
```

Questa funzione può anche risultare utile per **convertire un array di elementi DOM** (che non è un "vero e proprio" array):

```
var elementi = document.getElementsByTagName("div");
//elementi.constructor == Object

$.makeArray(elementi);

//elementi.constructor == Array
```

Infine, per verificare se una variabile è un vero array possiamo usare `$.isArray()`:

```
$.isArray([0,1,2]); // restituisce true
```

Versione originale: http://javascript.html.it/guide/lezione/4420/lavorare-con-gli-array/

Lavorare con gli oggetti

Per quanto riguarda gli oggetti, le due funzioni offerte da jQuery sono **each** e **extend**.

each

$.each() permette di definire operazioni da compiere ciclicamente su tutti gli elementi di una collezione, è molto simile a `$.map()`, ma accetta sia array che oggetti JavaScript come primo argomento.

Nel caso che segue vediamo che la funzione interna ha come argomenti le chiavi e i valori di un oggetto. Inoltre è possibile interrompere il ciclo inserendo un `return false`:

```
var oggetto = {"uno":"primo","due":"secondo","tre":"terzo"}

var oggettoFinale = {}; //un oggetto vuoto

$.each(oggetto,function (chiave,valore) {
    oggettoFinale[chiave]    = valore;
    if (chiave == "due") {
        // esce alla chiave 2
        return false;
    }
});

//oggettoFinale = {"uno":"primo","due":"secondo"}
```

extend

$.extend() estende l'oggetto passato come primo parametro con le proprietà del secondo. È importante ricordare è che la funzione modifica direttamente il primo oggetto e *sovrascrivendone le chiavi*:

```
var oggetto1 = {"uno":"primo","due":"due"};
var oggetto2 = {"due":"secondo"};

// oggetto1.due == "due"
$.extend(oggetto1,oggetto2);

// oggetto1.due == "secondo"
```

Per essere sicuri di non modificare, sovrascrivendole, le proprietà degli oggetti pre-esistenti è necessario creare un nuovo oggetto passando come primo argomento un oggetto vuoto:

```
var oggetto1 = {"uno":"primo","due":"due"};
var oggetto2 = {"due":"secondo"};

var oggetto3 = $.extend({},oggetto1,oggetto2);

// oggetto1.due == due
// oggetto3.due == secondo
```

Serializzare gli elementi

Come visto nella lezione su AJAX è possibile passare dei dati da inviare al server sotto forma di oggetto o stringa serializzata del tipo `"chiave=valore&..."`.

A questo scopo si può utilizzare `$.param()`, che accetta sia un oggetto JavaScript che una collezione di elementi form, i cui dati verranno serializzati rispettivamente nelle forme `chiave=valore` e `nomecampo=valore`.

```
var oggetto = {"uno":"primo","due":"due"};

var stringa = $.param({"uno":"primo","due":"due"});

//stringa = "uno=primo&due=secondo"
```

Lavorare sulle stringhe

L'unica funzione introdotta in jQuery per manipolare le stringhe è **$.trim()**, con la quale gli spazi all'inizio ed alla fine della stringa vengono eliminati:

```
$.trim("  stringa   ");
// restituisce "stringa"
```

Lavorare sulle funzioni

A differenza di altri framework, jQuery fornisce un'unica utility da utilizzare sulle funzioni, `$.isFunction()`, che restituisce `true` o `false` se l'argomento passato è una funzione o meno.

Versione originale: http://javascript.html.it/guide/lezione/4421/lavorare-con-gli-oggetti/

Browser e Feature Detection

Una delle funzioni più diffuse nelle librerie JavaScript e più usata dagli sviluppatori è il *browser detection* (o *browser sniffing*), cioè la possibilità di determinare quale browser stia utilizzando l'utente. In jQuery la funzionalità viene delegata alle proprietà dell'oggetto `$.browser`. In particolare viene impostata su `true` la proprietà relativa al browser in uso. Ecco i valori che troveremo se l'utente usa Internet Explorer:

```
$.browser.msie == true
$.browser.safari == false
$.browser.opera == false
$.browser.mozilla == false //Mozilla e Firefox
```

Per raffinare ulteriormente la ricerca è possibile ricavare la versione del browser in uso dalla proprietà `$.browser.version`. Riprendendo l'esempio precedente, ipotizziamo che il browser sia Internet Explorer 7:

```
$.browser.msie == true
$.browser.version == "7.0"
```

Una particolarità dell'oggetto `$.browser` è che viene reso disponibile **prima del caricamento del DOM**, rendendolo utile per associare eventi onDOMReady solo per certi browser:

```
if ($.browser.msie) {
    $(function () {
        alert("DOM caricato in IE!");
    });
}
```

In realtà dalla versione 1.3 l'uso di questo oggetto è sconsigliato in favore di `$.support` che, piuttosto del tipo di browser, determina quali siano le **caratteristiche attive o disponibili sul browser dell'utente**.
Questo cambiamento di rotta è legato alla constatazione che, la maggior parte delle volte, il *browser detection* serve a determinare il supporto o meno di certe caratteristiche in un browser, oppure ad attivare *workaround* per specifici bug.
Ricercare direttamente queste proprietà rende quindi tutto più diretto e soprattutto slega il codice dagli errori derivanti dall'avvento di nuove versioni dei browser.

Ecco in sintesi le proprietà disponibili (impostate su `true` se presenti):

- `boxModel`: il browser rispetta il W3C CSS Box Model? (falso in IE 6 e 7 in Quirks Mode, equivale a `$.browser.boxModel` in jQuery 1.2)
- `cssFloat`: è possibile accedere al valore CSS float con `style.cssFloat` (falso in IE)
- `hrefNormalized`: il risultato di `.getAttribute("href")` è lasciato intatto? (falso IE)
- `htmlSerialize`: gli elementi link sono serializzati correttamente con `.innerHTML`? (falso in IE)
- `leadingWhitespace`: gli spazi vuoti iniziali sono preservati con `.innerHTML`? (falso in IE 6-8)
- `noCloneEvent`: quando un elemento viene duplicato gli eventi collegati NON sono clonati? (falso in IE)
- `objectAll`: lanciando `.getElementsByTagName("*")` su un elemento restituisce tutti i suoi discendenti (falso in IE 7 e 8)
- `opacity`: esiste la proprietà CSS opacity? (falso in IE)
- `scriptEval`: Gli script vengono eseguiti se sono inseriti nel DOM con `.appendChild()` o `.createTextNode()` (falso in IE)
- `style`: `.getAttribute("style")` legge lo stile inline di un elemento? (falso in IE)
- `tbody`: il browser accetta una tabella senza un elemento `tbody`? (falso in IE)

Nelle future versioni di jQuery `$.support` rimarrà l'unico mezzo per determinare il browser dell'utente ed il suo supporto a determinate caratteristiche. Se proprio lo volessimo usare per identificare Internet Explorer 6 potremmo comunque scrivere:

```
if (!$.support.cssFloat && $.support.objectAll) {
        //codice per IE < 7
}
```

Versione originale: http://javascript.html.it/guide/lezione/4422/browser-e-feature-detection/

Internals e Data storage

Una parte interessante di jQuery riguarda gli **Internals** (funzioni interne) e il **Data Storage** (raccolta di dati).

Questi due argomenti sono strettamente legati poiché rappresentano un sistema interno a jQuery per aggiungere e ricavare dati da un elemento aumentandone le possibili proprietà oltre i semplici attributi. Prendiamo ad esempio l'elemento seguente:

```
//elemento di riferimento
<div class="miaClasse" id="mioId"></div>
```

dell'elemento `div` qui sopra possiamo ricavare classe e id sui quali lavorare:

```
var classe = $("div").attr("class");
var id = $("div").attr("id");
```

Se volessi aggiungere informazioni precise, per esempio il tipo di dato che può contenere, dovremmo aggiungerlo nella classe, oppure ricorrere ad un altro attributo fra quelli permessi:

```
<div class="miaClasse numeri" id="mioId"></div>
<div class="miaClasse" rel="numeri" id="mioId"></div>
```

In ambedue i casi saremmo ben presto limitati per numero di attributi disponibili e per le righe di codice necessario a ricavare i dati *di configurazione*.
Una strada per aggirare il problema è l'utilizzo del plugin metadata (http://plugins.jquery.com/project/metadata), che permette di inserire nella stringa dell'attributo `class` un oggetto JavaScript ricavabile poi con un metodo specifico:

```
<div class="miaClasse {type:'numeri'}" id="mioId"></div>

var data = $(".miaClasse").metadata();

  //data.type == 'numeri';
```

Il problema di questo approccio è che, oltre a caricare un nuovo file JavaScript nella pagina, *sporca* l'HTML con testo non direttamente correlato alla rappresentazione della pagina.

Data Storage

Anche se spesso non c'è alternativa al problema, una soluzione più corretta sarebbe utilizzare il metodo `.data()`:

```
<div class="miaClasse" id="mioId"></div>

//imposto il dato
$(".miaClasse").data("type","numeri");

var tipo = $(".miaClasse").data("type");
//tipo == 'numeri';
```

Da notare che se il dato non è impostato, il metodo restituisce `undefined`. Inoltre per rimuovere un dato basta passarne il nome al metodo `.removeData()`:

```
$(".miaClasse").removeData("type");
```

```
var tipo = $(".miaClasse").data("type");
//tipo == undefined;
```

Internals

Come alternativa al metodo `.data()` è possibile utilizzare le funzioni **Internals**, così chiamate perché utilizzate principalmente all'interno di `.data()` o più in generale del framework jQuery. Per questo motivo sono utlizzate raramente dagli sviluppatori, se non durante lo sviluppo di plugin ed estensioni del framework stesso.

Queste funzioni si comportano praticamente allo stesso modo dei metodi del *Data Storage*, ma naturalmente necessitano di un primo argomento che rappresenti un riferimento all'elemento DOM su cui lavorare:

```
$(".miaClasse").data("type","numeri");

//Equivale a
$.data( $(".miaClasse").get(0), "type", "numeri");
```

Da notare che **non è possibile passare alla funzione un oggetto jQuery**, ma bisogna estrarre un elemento singolo dalla collezione. Questa regola vale anche per le altre funzioni *Internals*.

Rispetto al *Data Storage*, sono disponibili inoltre due funzionalità aggiuntive:

- `$.data(elemento)`: restituisce un numero che rappresenta un'**ID univoco dell'elemento**
- `$.removeData(elemento)`: elimina **tutti i dati associati all'elemento** (usare `$.removeData(elemento,nomedato)` per rimuoverne solo uno)

Versione originale: http://javascript.html.it/guide/lezione/4423/internals-e-data-storage/

Aggiungere selettori

Uno dei maggiori punti di forza di jQuery è la facilità con la quale è possibile estendere le sue funzionalità. Questo può avvenire in tre modi:

- aggiungendo uno **pseudo-selettore** personalizzato al motore CSS
- aggiungendo un **metodo statico** (o funzione)
- aggiungendo un **metodo** (i cosiddetti *plugins*)

Aggiungere selettori

Come visto nella lezione sui selettori, jQuery offre alcuni pseudo-selettori che non sono rintracciabili nelle specifiche CSS come `:hidden` o `:checked`, ma sono stati introdotti dagli sviluppatori come scorciatoie per ricerche molto frequenti.

Nonostante il numero e le caratteristiche di questi pseudo-selettori riescano a coprire la maggior parte delle esigenze di scripting, potremmo voler rendere più snelle alcune operazioni (con conseguente riduzione del codice) aggiungendo degli pseudo-selettori personalizzati.

Tutto ciò è possibile estendo l'oggetto `$.expr[':']` con il metodo statico `$.extend()`già visto nella lezione sulle utlilty:

```
$.extend($.[":"], {

        nomeselettore : function (elemento) {
                //codice
        }

});
```

In sintesi, la funzione `nomeselettore` andrà a creare lo pseudo-selettore `$(":nomeselettore")` che opererà da filtro fra gli elementi del DOM e. Come nel metodo `.filter()`, `nomeselettore` dovrà restituire `true` affinché l'elemento sia selezionato.

Per esempio, ecco com'è realizzato lo pseudo-selettore `:checked`:

```
$.extend($.[":"], {

        checked : function (elem) {
                 //se l'elemento è selezionato restuisce true
                return elem.checked === true;
        }

});
```

Nell'esempio precedente è stata utilizzata come filtro una proprietà nativa dei campi di input JavaScript, tuttavia è anche possibile **utilizzare metodi propri di jQuery**.

Nel prossimo esempio vedremo di ricavare tutti i campi di input in un modulo che siano stati selezionati o compilati:

```
$.extend($.expr[':'],{
    filled : function(elem) {
```

```
        return (
                                $('[name='+elem.name+']:radio:checked').length == 1 ||
                                $(elem).is(':checkbox:checked') ||
                                ( $(elem).is('select,:text,textarea') && $(elem).val() != "" )
                        );
    }
});;
```

Come potete vedere, è possibile realizzare pseudo-selettori molto complessi, ma sicuramente molto efficaci e capaci di semplificare il nostro codice in operazioni ripetitive:

```
if ($(".richiesto").not(":filled").length > 0) {
                alert("Non tutti i campi richiesti sono stati compilati!");
} else {
                alert("I campi obbligatori sono stati compilati!");
}
```

Ecco questo nuovo pseudo-selettore in azione (http://www.html.it/guide/esempi/jquery/esempi/14-estendere-1.html).

Versione originale: http://javascript.html.it/guide/lezione/4471/aggiungere-selettori/

Aggiungere metodi

Nella lezione sulle Utility abbiamo visto come jQuery non offra un gran numero di metodi statici (o funzioni), confidando nella capacità dello sviluppatore di utilizzare i metodi nativi JavaScript. Vi sono alcuni casi, tuttavia, in cui utilizzare JavaScript nativo potrebbe rendere ripetitivo il nostro codice. Ecco che allora è possibile aggiungere nuovi metodi statici sfruttando il fatto che se a `$.extend()` viene passato un solo argomento **ad essere esteso è l'oggetto jQuery stesso**.

Come esempio prendiamo il caso di voler realizzare una funzione per rimuovere gli spazi in eccesso da una stringa, non solo all'inizio e alla fine (come fa `$.trim()`, ma anche fra parola e parola. Ecco il codice necessario:

```
$.extend({
        clean : function (stringa) {
                //sostituisce le porzioni di stringa con più di uno spazio
                //con un singolo spazio
                var str = stringa.replace(/\s+/g, ' ');

                //quindi restituisce la stringa eleminando eventuali spazi iniziali e finali
                return $.trim(str);
        }
});
```

Per applicare la nuova funzione basterà passarle la stringa da processare:

```
var stringa = "  stringa   piena     di  spazi";
$.clean(stringa);
//stringa = "stringa piena di spazi"
```

Aggiungere un metodo

L'ultima possibilità di estensione prevista nativamente da jQuery è quella di aggiungere un metodo, realizzando un cosiddetto ***plugin***. Il codice necessario a realizzare un'estensione di questio tipo può essere molto semplice come molto complesso, in base alle funzionalità che si vogliono realizzare. In generale il punto di partenza per la creazione di un plugin è la funzione `$.fn.extend()`. Ecco lo schema dal quale partire:

```
$.fn.extend({

  nomeplugin : function () {

          return this.each(function () {
                        //this è l'elemento
          });

        }

});
```

Nell'esempio viene passato un oggetto a `$.fn.extend` contentente il metodo da aggiungere a jQuery. Al suo interno il `return this...` permette di restituire un riferimento all'oggetto jQuery, preservando l'eventuale concatenabilità del nuovo metodo, mentre ricorrendo a `.each()` si potrà ciclare fra i singoli elementi della collezione.
Da notare che se all'interno della funzione passata ad `.each()` verrà usato `return`, allora il plugin restituirà un

valore, perdendo la concatenabilità.

Per chiarire questa procedura, ecco il codice necessario per realizzare un semplice plugin che imposti su giallo il colore di sfondo di tutti gli elementi della collezione:

```
$.fn.extend({

        evidenzia : function () {
                return this.each(function () {
                        $(this).css("backgroundColor","yellow");
                });
        }

});
```

Per richiamare il plugin basterà scrivere:

```
$("p").evidenzia();
```

Ecco il plugin in funzione (http://www.html.it/guide/esempi/jquery/esempi/14-estendere-2.html).

Grazie a questo approccio possono essere realizzati plugin molto complessi in grado di gestire, per esempio, gallerie di immagini e finestre modali. jQuery offre già un buon numero di plugin da scaricare per approfondire le possibilità di estensione del framework a questo indirizzo: http://plugins.jquery.com/ (http://plugins.jquery.com/).

Versione originale: http://javascript.html.it/guide/lezione/4472/aggiungere-metodi/

Usare gli alias

Nelle prime lezioni abbiamo visto che l'oggetto `$` altro non è che un alias di `jQuery` creato per comodità e per convenzione. Abbiamo anche visto che è possibile, per questioni di compatibilità con altre librerie, cancellare questo alias con `jQuery.noConflict()`. Una volta lanciata questa funzione, **non sarà più possibile** richiamare una collezione con `$("p, div")`, ma dovremo usare `jQuery("p, div")`.
Un problema che potrebbe insorgere, allora, è che riferendoci a `$` all'interno delle nostre estensioni e dei nostri script, questo riferimento non venga trovato o addirittura sia già associato ad altre librerie. Questo è ancora più vero quando decidiamo di rilasciare le nostre estensioni ad altri sviluppatori, con la conseguente necessità di non interferire con i loro script.

Per aggirare il problema è buona norma utilizzare la tecnica delle closure con una funzione *auto-eseguente*, in questo modo:

```
(function ($) {

 $.extend({
     nuovafunzione : function () {

     }
 });

})(jQuery);
```

Il codice qui sopra definisce inizialmente una funzione anonima con un argomento `$`. Nell'ultima riga la funzione anonima viene eseguita passandogli come argomento l'oggetto jQuery, che sarà rappresentato all'interno della funzione da `$`. Questo ci permetterà di mantenere un comodo riferimento a jQuery senza troppi pensieri, oppure addirittura di definire una nuova scorciatoia che comunque avrà valore solo all'interno della funzione anonima:

```
(function (jq) {

 jq.extend({
     nuovafunzione : function (variabile) {
                jq.makeArray(variabile)
                        //.......
     }
 });

})(jQuery);
```

Conclusione

Con questa lezione si conclude la guida a jQuery. Senza dubbio ad un primo sguardo il gran numero di metodi offerti potrebbe disorientare chi si avvicina per la prima volta ad un framework JavaScript, tuttavia, con un po' di pratica, sarà difficile far a meno della sinteticità e dell'affidabilità di questa libreria. Ecco una lista di link (in inglese) utili per tutti gli approfondimenti e per un riferimento rapido sui metodi disponibili:

- Documentazione Ufficiale jQuery (http://docs.jquery.com/Main_Page)
- Tutorial su jQuery (http://docs.jquery.com/Tutorials)
- Creare un plugin con jQuery (http://docs.jquery.com/Plugins/Authoring)
- Domande frequenti su jQuery (http://docs.jquery.com/Frequently_Asked_Questions)

Versione originale: http://javascript.html.it/guide/lezione/4473/usare-gli-alias/