Guida a jQuery

Gabriele Romanato

http://gabrieleromanato.com/

Plugin jQuery: come crearli

I plugin di jQuery non sono altro che estensioni aggiunte alla libreria stessa. I plugin vanno ad operare sugli elementi del DOM e sugli oggetti di un set jQuery esattamente come qualsiasi altro metodo nativo di jQuery (come `addClass()` ad esempio). La particolarità dei plugin è che funzionano proprio come i metodi nativi di jQuery. Essi operano all'interno del namespace jQuery e ogni plugin va a creare un suo namespace particolare all'interno del namespace globale della libreria. Possono funzionare sia su elementi HTML particolari che su più elementi. In questo post vedremo come creare un plugin da zero. Scopo del nostro plugin sarà quello di aggiungere alcuni widget di Facebook agli elementi della pagina.

Creazione del namespace

Per creare il namespace per il nostro plugin è sufficiente il seguente codice:

```
(function($) {    // codice del plugin       })(jQuery)
```

Come si può notare, si tratta di una funzione di tipo self-executing (che si esegue senza bisogno di essere invocata) avente come parametri l'oggetto `jQuery` stesso. Quindi il codice del nostro plugin opererà proprio all'interno di jQuery, ossia ne costituirà un'estensione.

Uso dell'oggetto fn

Ciascun oggetto JavaScript ha un oggetto `prototype` che viene creato con l'oggetto stesso. Tale oggetto serve per gestire l'ereditarietà e per estendere l'oggetto base con nuovi metodi e proprietà. jQuery non fa eccezione: l'oggetto `fn` è un alias dell'oggetto`prototype` della libreria. Tramite tale oggetto, possiamo estendere jQuery con il nostro plugin definendo una semplice funzione, ossia aggiungendo un metodo a jQuery che funzionerà esattamente come i suoi metodi nativi:

```
(function($) {    $.fn.FacebookSocial = function(options) {             // codice del
plugin             };   })(jQuery);
```

Quindi se ora abbiamo un elemento con ID uguale a `#test` possiamo scrivere:

```
$('#test').FacebookSocial(options);
```

Infatti il nostro plugin ora può funzionare su tutti gli elementi HTML selezionabili da jQuery. Il parametro `options` (che può anche essere chiamato `settings` o in modo simile) andrà a contenere i valori di default delle opzioni del nostro plugin, ossia quei valori che il plugin userà se non ne vengono specificati altri all'atto del suo uso. Vedremo il suo funzionamento di seguito.

Opzioni predefinite

Le opzioni predefinite per un plugin si rivelano fondamentali nel caso si decidesse di usarlo senza opzioni, ovvero invocandolo in questo modo:

```
$('#test').FacebookSocial();
```

Senza dei valori predefiniti, il plugin non funzionerebbe. Quindi andremo a definire un nuovo oggetto letterale contenente i valori di default e quindi estenderemo (usando `$.extend()`) `options` con questo oggetto letterale. Il risultato finale sarà che `options`diverrà un oggetto letterale contenente i nostri valori predefiniti:

```
(function($) {    $.fn.FacebookSocial = function(options) {          var defaults = {
                 header: false,                colorscheme: 'light',
recommendations: false,                transparency: false          };
```

L'oggetto this

All'interno di un plugin jQuery, `this` fa sempre riferimento all'oggetto jQuery (o all'elemento del DOM) su cui si sta operando. Quindi se abbiamo:

```
$('#test').FacebookSocial();
```

`this` fa riferimento all'elemento con ID uguale a `#test`. Tutti i plugin di jQuery devono restituire `this` tramite un'istruzione `return` in modo da consentire il normale proseguimento della catena di jQuery e della cascata, ossia per avere:

```
$('#test').FacebookSocial(). addClass('test').hide();
```

Per migliorare la performance, possiamo memorizzare `this` in una variabile e usarla in seguito:

```
(function($) {    $.fn.FacebookSocial = function(options) {          var defaults = {
                 header: false,                colorscheme: 'light',
recommendations: false,                transparency: false          };
});             };   })(jQuery);
```

Il metodo `each()` viene usato in questo contesto per permettere un'azione distribuita su un set composto da più elementi (per esempio aventi una determinata classe CSS).

Interagire con le opzioni

Abbiamo detto che l'oggetto letterale `options` ora contiene i valori di default delle opzioni del nostro plugin. Essendo un oggetto, possiamo accedere alle sue proprietà tramite la notazione `oggetto.proprietà` e quindi usarle per far compiere al nostro plugin determinate azioni:

```
(function($) {    $.fn.FacebookSocial = function(options) {          var defaults = {
                header: false,                colorscheme: 'light',
recommendations: false,                transparency: false          };
{
        case 'like':
var iframe = '<iframe src="http://www.facebook.com/widgets/like.php?href=' + options.url +

                                                                    case 'sharer':
title="Add to Facebook">' + options.text + '</a>';
$(options.wrapper).html(link).appendTo(that);
src="http://www.facebook.com/plugins/activity.php?site=' + options.url +
options.recommendations + '" class="' + options.klass + '" scrolling="no" frameborder="0"
allowTransparency="' +                                    options.transparency + '" />';
                                                                    default:
});             };   })(jQuery);
```

Il plugin finito

Potete vedere il plugin in azione qui e scaricarlo a questo indirizzo.

jQuery: il metodo data()

Il metodo data() di jQuery permette di associare dati arbitrari a ciascun oggetto del set di jQuery. Questi dati possono essere memorizzati come oggetti letterali. L'utilità di questo metodo sta nel fatto che migliora di molto la performance dei nostri script, in quanto i dati inizializzati vengono direttamente associati agli elementi.

Il primo modo per associare tali dati è quello di memorizzarli in modo semplice:

```
$('#test').data('test', 10);
```

Possiamo quindi accedere al loro valore in questo modo:

```
alert($('#test').data('test')); // 10
```

Per gli oggetti complessi, invece, si usa la seguente notazione:

```
$('#test').data('myTest', {foo: 'Foo', bar: 10});
```

Per accedere ai valori di tali oggetti useremo invece:

```
alert($('#test').data('myTest').foo); // 'Foo' alert($('#test').data('myTest').bar); // 10
```

In altre parole, `test` è divenuto il riferimento per accedere all'oggetto in esso contenuto. Come si può notare, la notazione è la stessa di qualsiasi altro oggetto JavaScript.

Possiamo ottenere un riferimento completo a tutti i dati associati all'elemento usando `data()` senza argomenti:

```
$('#test').data(); // {test: 10, myTest: {foo: 'Foo', bar: 10}}
```

Una particolarità interessante sta nel fatto che possiamo anche specificare metodi (cioè funzioni) per gli oggetti associati agli elementi:

```
$('#test').data('myTest', {method: function() { alert('Test');}});
```

e quindi invocare tali metodi nel modo consueto:

```
$('#test').data('myTest').method(); // un alert con 'Test'
```

jQuery: animazioni

In questo articolo vorrei illustrare i princìpi alla base delle animazioni di jQuery. Sia che si tratti di metodi che operano su animazioni particolari (come `fadeIn()`) o su animazioni più complesse (come `animate()`), essi sono tutti accomunati da caratteristiche comuni rintracciabili nell'oggetto `fx` di jQuery. Questo oggetto è direttamente accessibile dagli sviluppatori. Non è un caso che plugin come Easing lo utilizzino per estendere la gamma degli effetti disponibili in jQuery. Vediamo di seguito le caratteristiche principali delle animazioni jQuery.

Gli step

Ogni animazione jQuery è suddivisa in atomi di tempo chiamati step. Ciascuno step può anche essere considerato come l'unità di misura base di un animazione. Quando creiamo un'animazione, viene automaticamente creata una sequenza di step che porterà jQuery a variare una proprietà CSS dell'elemento dallo stato iniziale a quello finale.

Quando scriviamo:

```
$('#test').animate({   width: 200 });
```

jQuery legge la proprietà `width` iniziale dell'elemento (per esempio 0) e crea la sequenza di step che porterà l'elemento ad avere la larghezza finale di 200 pixel. In questo caso, poichè non è stata specificata una velocità per l'animazione, jQuery userà il valore`fast` come valore iniziale.

A questo punto jQuery crea la sequenza di step rapportandola con la velocità specificata in modo da creare l'effetto di una maggiore o minore velocità dell'animazione. Un'animazione con un parametro di velocità inferiore ha più step di un'animazione che ha un parametro di velocità più elevato.

Gli step servono anche per tener traccia dell'inizio e della fine dell'animazione. Possiamo accedere a tali step usando il secondo oggetto letterale del metodo `animate()`:

```
$('#test').animate({   width: 200 }, {   step: function() {          //...      } } );
```

A scopo di test potremmo inizializzare un contatore e incrementarlo su ogni step. Potremo così vedere di fatto il numero di step utilizzati in un'animazione:

```
var counter = 0;  $('#test').animate({   width: 200 }, {   step: function() {
counter++;     console.log(counter);      } } );
```

Il metodo `step()` non ha un'utilità immediata. Osservando cosa accade nella console JavaScript del browser, noteremo una elevata velocità del nostro contatore che non corrisponde ovviamente alla velocità globale dell'animazione. In altre parole, step e velocità funzionano in modo asimmetrico.

L'animazione termina quando la sequenza di step ha portato il valore iniziale al suo stato finale (nel nostro esempio da 0 a 200).

Animazioni sequenziali e animazioni simultanee

Come abbiamo visto, un'animazione jQuery non è altro che una sequenza di step che modifica un valore portandolo da uno stato iniziale ad uno finale. Tuttavia molti metodi di jQuery che gestiscono animazioni accettano una funzione di callback da invocare quando l'animazione è completa.

Possiamo utilizzare tali funzioni per creare animazioni sequenziali:

```
$('#test').animate({   width: 200 }, 'slow', function() {    $(this).animate({     left:
100        }, 'slow', function() {        $(this).animate({       width: 100     },
'slow');      });  });
```

Ossia:

1. l'elemento raggiunge la larghezza di 200 pixel
2. viene spostato verso destra di 100 pixel
3. la sua larghezza viene reimpostata a 100 pixel

Queste animazioni sono sequenziali. Un'animazione simultanea sarebbe invece stata la seguente:

```
$('#test').animate({   width: 200,   left: 100 }, 'slow');
```

In questo caso le due proprietà CSS vengono modificate simultaneamente.

Animazioni ripetute

Le animazioni jQuery possono essere ripetute nel tempo. Per far ciò occorre utilizzare le funzioni `setTimeout()` o `setInterval()` di JavaScript. Possiamo utilizzare `setInterval()` in questo modo:

```
var interval = setInterval(function() {      $('#test').animate({      width: 200    },
500, function() {            $(this).animate({        width: 100      }, 500);        });
}, 1000);
```

L'unico accorgimento da usare in questi casi riguarda il tempo delle animazioni, che deve sempre coincidere con la durata del timer JavaScript. In questo caso il nostro timer ha una durata di 1000

millisecondi, quindi ciascuna animazione dovrà durare 500 millisecondi (o un altro valore la cui somma restituisca 1000).

jQuery e classi CSS

jQuery gestisce le classi CSS attraverso i metodi `addClass()`, `removeClass()` e `toggleClass()`. Il primo metodo aggiunge una classe CSS, il secondo la rimuove e il terzo alterna la classe corrente con un'altra classe passata come parametro operando uno switch sulla classe. Tuttavia, la prima cosa da capire per gestire correttamente le classi CSS in jQuery è la cascata e la specificità dei CSS. Vediamo i dettagli.

Cascata e specificità

Una classe più specifica sovrascrive una classe meno specifica secondo la cascata quando si hanno regole in conflitto. Per esempio:

```
.classe1 {   background: red; }  p.classe2 {   background: green; /* vincitrice */ }
```

La seconda classe ha un selettore di tipo, quindi è più specifica. Nel caso di identica specificità, vince la classe che viene dopo nel sorgente.

```
.classe1 {   background: red; }  .classe2 {   background: green; /* vincitrice */ }
```

Quando invece le classi non hanno regole in conflitto, gli stili risultanti sono la somma degli stili delle singole classi.

addClass()

Il metodo `addClass()` accetta uno o più nomi di classe come parametri e li aggiunge all'elemento:

```
$('#add').click(function(event) {        $('#test').addClass('add');
event.preventDefault();       });
```

In questo caso il metodo aggiunge la classe `add`:

```
.add {   background: #ffc;   width: 150px; }
```

removeClass()

Il metodo `removeClass()` accetta uno o più nomi di classe come parametri e li rimuove dall'elemento:

```
$('#remove').click(function(event) {        $('#test').removeClass($(this).attr('class'));
event.preventDefault();       });
```

In questo caso il metodo rimuove la classe impostata sull'elemento.

toggleClass()

Il metodo `toggleClass()` scambia il nome della classe corrente con quello della classe passata come parametro:

```
$('#toggle').click(function(event) {        $('#test').toggleClass('toggle');
event.preventDefault();       });
```

In questo caso il metodo usa la seguente classe CSS:

```
.toggle {   background: #c00;   color: #fff;   padding: 1em;   font-weight: bold;   width:
100px; }
```

Potete visionare gli esempi in questa pagina.

jQuery: $.grep() e $.map()

I metodi globali di jQuery `$.grep()` e `$.map()` operano entrambi sugli array, ma con un'importante differenza: il primo filtra un array e restituisce i valori filtrati in un nuovo array, mentre il secondo applica una funzione ad ogni elemento dell'array e restituisce un array modificato. Vediamo in dettaglio queste differenze.

$.grep()

La sintassi d'uso di questo metodo è la seguente:

```
$.grep(array, function(index, value) {    //...  });
```

Il primo parametro è l'array che vogliamo filtrare. `index` è l'indice numerico di ciascun elemento dell'array, mentre `value` è il valore di ciascun elemento.

$.map()

La sintassi d'uso di questo metodo è la seguente:

```
$.map(array, function(value, index) {    //...  });
```

Il primo parametro è l'array a cui vogliamo applicare una funzione su ciascuna voce. `value` è il valore di ciascun elemento nell'array, mentre `index` è il suo indice numerico.

Uso pratico

Abbiamo una lista non ordinata con varie voci. Ciascuna voce contiene un numero progressivo. Usando `$.grep()` possiamo ad esempio creare un nuovo array che contenga solo le voci con numeri pari:

```
var lis = $('#test li').get(); var grep = $.grep(lis, function(index, value) {      return
(lis[value].firstChild.nodeValue % 2 == 0);       });
```

L'istruzione `return` serve in questo caso ad applicare il nostro filtro.

Se invece vogliamo modifcare l'array e restituire un array modificato, possiamo usare `$.map()`. Per esempio, possiamo sommare il numero contenuto in ciascuna voce all'indice numerico di ciascun elemento:

```
var map = $.map(lis, function(value, index) {        return ('<li>' + (new
Number(value.firstChild.nodeValue) + index) + '</li>');  });
```

Come si può notare dai due esempi, i nuovi array saranno contenuti nelle variabili `grep` e `map` associate con i rispettivi metodi.

jQuery: creare elementi

jQuery permette di creare elementi al volo usando il wrapper `$()`. Questo metodo accetta sia stringhe che espressioni DOM, e si rivela incredibilmente potente per alterare la struttura di un documento con estrema precisione e affidabilità. Vediamone insieme i dettagli.

Stringhe

Si possono usare stringhe come la seguente:

```
$('<div class="test" id="test"/>').text('Test').appendTo('body');
```

o si possono passare attributi e metodi dell'elemento in un oggetto letterale usato come secondo argomento:

```
$('<div/>', {   'class': 'test',   id: 'test',   text: 'Test' }).appendTo('body');
```

Si noti come `class` vada racchiusa tra virgolette in quanto parola riservata ECMAScript.

Espressioni DOM

Si possono anche usare espressioni DOM come la seguente:

```
$(document.createElement('div')). addClass('test'). attr('id', 'test'). text('Test').
appendTo('body');
```

jQuery converte automaticamente il riferimento restituito dal metodo DOM in un nuovo elemento del suo set.

jQuery: step e animazioni

Ho appena pubblicato un esempio in jQuery che illustra le possibilità offerte dagli step delle animazioni jQuery. Uno step è un singolo componente atomico di un'animazione jQuery. Un'animazione può contenere diversi step, che variano a seconda del tipo di animazione creata e dei suoi parametri. Sostanzialmente, uno step viene gestito internamente dall'oggetto `fx` di jQuery che lo usa per aggiungere ritardi, accelerazioni e easing. Il metodo `animate()` fornisce a sua volta il

metodo `step()` tra le sue opzioni che ci consente di eseguire un'azione su ogni step. Vediamo un esempio pratico.

Il metodo `animate()` di norma viene usato in questo modo:

```
$(elemento).animate({   left: 100 }, 800, function() {    console.log('Completato');  });
```

Ma si può anche usare la sintassi estesa utilizzando il secondo oggetto letterale per gestire le sue opzioni:

```
$(elemento).animate({   left: 100 }, {   duration: 800,      complete: function() {
console.log('Completato');      } } );
```

È proprio in questo oggetto letterale che possiamo usare il metodo `step()`:

```
$(function() {   $('div', 'body').click(function() {      var width = $(this).width();
var height = $(this).height();      $(this).animate({      left: 100    }, {      step:
function() {              $(this).width(width -= 1);        $(this).height(height -= 1);
},            complete: function() {              var width2 = $(this).width();        var
height2 = $(this).height();              $(this).animate({                  left: 0
},          {                  step: function() {                        $(this).width(width2
+= 1);            $(this).height(height2 += 1);                                    },
duration: 800                }                );            },            duration: 800
});        });   });
```

Nell'esempio le dimensioni dell'elemento sono state modificate di un'unità alla volta utilizzando gli step. Ogni step fa aumentare o diminuire le dimensioni dell'elemento.

Altre applicazioni

Gli step possono essere utilizzati per gestire meglio le animazioni simultanee e parallele su più elementi. In ogni step è possibile usare un riferimento ad un elemento diverso da quello selezionato e modificare il suo stato in modo dinamico.

jQuery: differenza fra position() ed offset()

La differenza tra i metodi `position()` e `offset()` di jQuery sta nel modo con cui vengono calcolate le coordinate restituite. Nel primo caso, le coordinate vengono calcolate sempre rispetto al genitore dell'elemento, mentre nel secondo caso tali coordinate possono essere calcolate rispetto all'offset dell'elemento genitore rispetto alla pagina quando il genitore è posizionato, ossia ha un valore CSS diverso da `static`. Se il genitore non è posizionato, entrambi i metodi resituiscono gli stessi valori. Vediamo un esempio.

Abbiamo la seguente struttura:

[xml]   [/xml]

e i seguenti stili:

```
#container {   width: 600px;  height: 400px; background: #ccc;      position: relative; }
#test {        width: 100px;  height: 100px; background: #d34545;   position: relative; }
```

In questo caso il genitore è posizionato. Se proviamo ad aggiungere 50 pixel ai valori delle proprietà `top` e `left` restituite da questi metodi, noteremo sicuramente la differenza:

```
$(function() {    var test = $('#test', '#container');   var triggerPosition =
$('#position');   var triggerOffset = $('#offset');
triggerPosition.click(function(event) {        test.css({top: 0, left: 0});        var top
= test.position().top;     var left = test.position().left;          test.animate({
top: top + 50,       left: left + 50     }, 'slow');          event.preventDefault();
});      triggerOffset.click(function(event) {        test.css({top: 0, left: 0});
var top = test.offset().top;     var left = test.offset().left;          test.animate({
top: top + 50,       left: left + 50     }, 'slow');          event.preventDefault();
});    });
```

Con il primo metodo otteniamo una distanza inferiore rispetto al secondo, pur avendo specificato gli stessi valori da aggiungere per entrambi. Questo accade perchè nel caso di `offset()` le coordinate sono il risultato della somma dell'offset del genitore rispetto alla pagina e del nostro valore. Potete visionare l'esempio finale in questa pagina.

jQuery: il metodo $.getScript()

Il metodo AJAX $.getScript() di jQuery ci consente di caricare script JavaScript e di eseguirli nel contesto globale. Tuttavia, questo metodo non restituisce errori qualora lo script da includere contenga errori sintattici, quindi per il debugging è necessario ricorrere all'evento `ajaxError`. Vediamo i dettagli.

Vogliamo caricare il seguente script:

```
var Library = {    create: function(options) {        options = {       element: '<p/>',
content: 'Test',       target: 'body'     };          $(options.element).
text(options.content).     appendTo(options.target);        },    init: function() {
this.create();      }   };
```

Quando lo script è stato caricato, vogliamo eseguire il metodo `init()` dell'oggetto `Library`. Prima però dobbiamo intercettare i possibili errori che potrebbero verificarsi:

```
$('#log').ajaxError(function(e, jqxhr, settings, exception) {          $(this).text(e + ' '
+ jqxhr + ' ' + exception);      });
```

Ovviamente potete più semplicemente inviare questi dati alla console JavaScript. Quindi possiamo usare `$.getScript()`:

```
$.getScript('jquery-get-script-test.js', function() {        Library.init();    });
```

Come si può notare, il metodo invocato viene eseguito come se l'oggetto a cui appartiene fosse già presente nella pagina. Potete visionare l'esempio finale in questa pagina.

jQuery: debugging degli elementi nella console

Quando si esegue il debugging di uno script jQuery è di fondamentale importanza tenere traccia degli elementi e della loro struttura. Una pratica raccomandata è quella di inviare dei messaggi alla console JavaScript del browser per ispezionare gli elementi, sia che essi siano stati creati dinamicamente sia che siano già presenti nella struttura del DOM. Vediamo come semplificare al massimo questo processo.

Possiamo creare il seguente plugin, `log()`:

```
(function($) {       $.fn.log = function() {              var that = this;         var
selector = that.selector;         var context = that.context;         var element =
that[0];         var attrs = element.attributes;         var name =
element.tagName.toLowerCase();         var contents = that.html();         var logContent =
'';                          return that.each(function() {
logContent += '[name]: ' + name;             logContent += ' [selector]: ' + selector;
logContent += ' [context]: ' + context;             logContent += ' [attributes]: ';
if(attrs.length > 0) {                              for(var i = 0; i < attrs.length; i +=
1) {                                  var attrName = attrs[i].name;                    var
attrValue = attrs[i].value;                                        logContent +=  attrName +
': ' + attrValue + ' ';                                                }
} else {                                 logContent += 'no attributes ';
}                          logContent += ' [contents]: ' + contents;
```

```
console.log(logContent);                                        });
};    })(jQuery);
```

Questo semplice plugin invia alla console JavaScript il nome dell'elemento, il suo selettore, il suo contesto, i suoi eventuali attributi con i rispettivi valori e i suoi contenuti HTML.

Vediamo un esempio. Data la seguente struttura:

[xml] <ul id="test"> <li id="a" class="a" title="a">A B <li id="c" class="c">C [/xml]

Possiamo usare il plugin in questo modo:

```
$(function() {    $('li', '#test').each(function() {        $(this).log();      });   });
```

E avremo il seguente output nella console:

[caption id="attachment_1071" align="aligncenter" width="700" caption="L'output del plugin log() nella console del browser"]

[/caption]

jQuery: personalizzare l'easing

jQuery ci permette di modificare anche il suo funzionamento interno consentendo l'accesso ai suoi oggetti core. Uno di questi oggetti è appunto `easing`, che controlla gli effetti di easing sulle animazioni. Questo oggetto può essere esteso definendo i nostri valori di easing personalizzati che prendono la forma di metodi di tale oggetto. Vediamo insieme i dettagli.

La sintassi di base è la seguente:

```
jQuery.extend(jQuery.easing, {                nomeEffetto: function(x, t, b, c, d) {
          return espressione;          }  });
```

I parametri che ci interessano del metodo che definiremo sono:

1. `t`: tempo corrente
2. `b`: valore iniziale
3. `c`: valore modificato
4. `d`: durata

Il metodo dovrà restituire un'espressione matematica che opera sui quattro parametri dell'animazione. Per esempio:

```
jQuery.extend(jQuery.easing, {                moveAway: function(x, t, b, c, d) {
return (c - b) * (d / t);             }   });
```

Quindi possiamo usare il nuovo easing `moveAway` con il metodo `animate()`:

```
$(function() {    $('#test').click(function() {        $(this).animate({      width: 200,
height: 200,      left: 200,      visibility: 'toggle'     }, 1000, 'moveAway');
});   });
```

Vi consiglio di studiare il codice sorgente del plugin Easing per rendervi conto dei principali valori di easing possibili e, soprattutto, per studiare le espressioni matematiche usate.

Potete visualizzare l'esempio finale in questa pagina.

jQuery: il ciclo each

Esistono due tipi di cicli `each()` in jQuery: il ciclo `each()` che opera sugli elementi di un set di jQuery e il ciclo `$.each()` che opera su oggetti e array. Pur essendo simili nel concept, questi metodi presentano delle differenze che vale la pena approfondire per evitare ogni possibile confusione.

Il ciclo each()

Il ciclo `each()` opera sugli elementi del set di jQuery. Accetta come parametro opzionale l'indice degli elementi esaminati. La sua sintassi è la seguente:

```
$(elementi).each(function(i) {    //...  });
```

Gli elementi del set vengono gestiti uno alla volta. L'indice (facoltativo) indica la posizione di un elemento nel set, mentre `$(this)` si riferisce all'elemento corrente:

```
$('p').each(function(i) {      var $p = $(this);              console.log(i + ': ' + $p);
        /*              0: riferimento a oggetto jQuery               1: riferimento a
oggetto jQuery          ....                    */                      });
```

Per uscire da questo tipo di ciclo si può:

1. usare l'espressione `return false`
2. usare un valore numerico da confrontare con l'indice del ciclo

Il primo caso è utile quando abbiamo trovato l'elemento del set che soddisfa una certa condizione:

```
$('p').each(function() {       var $p = $(this);      var text = $p.text();
        if(text.length < 250) {                       //...                          return
false;          }  });
```

Il secondo caso, invece, serve a restituire solo un dato numero di elementi:

```
$('p').each(function(i) {      //...           return (i < 5);  });
```

In questo caso verranno restituiti solo i primi sei elementi del set (l'indice inizia da 0).

Il ciclo $.each()

Questo ciclo opera su oggetti e array (o su strutture simili ad array). Accetta tre argomenti:

1. l'oggetto su cui operare
2. l'indice degli elementi dell'oggetto
3. il valore degli elementi dell'oggetto

La sua sintassi è la seguente:

```
$.each(oggetto, function(indice, valore) {    });
```

I parametri `indice` e `valore` hanno un diverso significato a seconda del tipo di oggetto passato come primo parametro:

1. in un array, essi si riferiscono all'indice numerico di ciascuna voce e al valore della voce stessa
2. in un oggetto, essi si riferiscono al nome della proprietà corrente dell'oggetto e al valore di tale proprietà

Esempio del primo caso:

```
var arr = ['a', 'b', 'c'];  $.each(arr, function(indice, valore) {   console.log(indice + ':
' + valore);           // 0: 'a' 1: 'b' 2: 'c'   });
```

Anche in questo caso l'indice inizia da zero. Un esempio del secondo caso:

```
var obj = {    a: 'A',        b: 'B',        c: 'C' };  $.each(obj, function(nome, valore)
{       console.log(nome + ': ' + valore);            // a: 'A' b: 'B' c: 'C'  });
```

Si può uscire da questo tipo di ciclo con le stesse modalità viste in precedenza per il ciclo `each()`. Un caso interessante è quando si utilizza questo ciclo sulle strutture DOM di tipo collection o list. In questo caso l'oggetto del DOM è sempre rappresentato dal terzo parametro (`valore`).

jQuery: trasformazione e controllo del DOM

Uno dei motivi per cui jQuery è forse la libreria JavaScript più popolare sul web sta nella sua capacità di dare allo sviluppatore la facoltà di trasformare e manipolare il DOM a suo piacimento. In altre parole, jQuery ci da un controllo totale sul DOM, permettendoci di creare strutture complesse a partire da strutture semplici. In questo articolo vedremo appunto come trasformare una struttura lineare in una struttura annidata.

Partiamo dalla seguente struttura semplice:

[xml] <h2>...</h2> ... <!--serie di paragrafi--> <h2>...</h2> ... <!--serie di paragrafi--> [/xml]

Vogliamo arrivare ad avere la seguente struttura:

[xml]   ... <a>... <form class="contact">...</form> <h2>...</h2> ... ... <!--serie di paragrafi--> <!--altri elementi con classe post--> [/xml]

Partiamo subito col suddividere la struttura di partenza in elementi con classe `post`:

```
$('h2', '#content').each(function() {         var $h2 = $(this);
        $h2.nextUntil('h2').andSelf().wrapAll('<div class="post"></div>');    });
```

Abbiamo usato i seguenti metodi:

1. `nextUntil()`: seleziona gli elementi successivi fino all'elemento specificato (non incluso)
2. `andSelf()`: include l'elemento di partenza nella selezione
3. `wrapAll()`: racchiude tutti gli elementi specificati in un elemento HTML definito come stringa

Quindi passiamo ad aggiungere gli elementi `date` e `author`:

```
var date = new Date(document.lastModified); var month = date.getMonth()+1; var day =
date.getDate(); var year = date.getFullYear();
$('div.post', '#content').each(function() {         var dt = $('<div
class="date"/>').html('<span class="day-month">' + day + ' ' + month +
'</span>' + '<span class="year">' + year + '</span></div>');         var $h2 = $('h2',
$(this));    dt.insertAfter($h2);         var author = $('<div
class="author"/>').html('<strong>Autore:</strong>' +
        '<a href="mailto:gabriele.romanato@gmail.com">Gabriele Romanato</a>');
```

Abbiamo usato i seguenti metodi:

1. `html()`: crea il contenuto HTML di un elemento usando la stringa passata come argomento
2. `insertBefore()`: inserisce un elemento prima di un altro
3. `insertAfter()`: inserisce un elemento dopo un altro elemento

Non ci resta quindi che inserire il form con classe `contact`:

```
$('div.author', '#content').each(function() {         var $author = $(this);    var form =
$('<form/>', {     action: '#',     method: 'post',     'class': 'contact'
}).html('<div>' +             '<label for="nome">Nome</label>' +                  '<input
type="text" name="nome"/>' +                  '<label for="email">E-mail</label>' +
for="messaggio">Messaggio</label>' +                 '<textarea name="messaggio" rows="15"
cols="15"/>' +                '<p><input type="submit" name="invia" value="Invia" /></p>'+
              '</div>');    form.appendTo($author);          });
```

Abbiamo usato il metodo `appendTo()`, che aggiunge un elemento all'elemento di destinazione.

Sintassi del wrapper $()

Come avrete notato, il wrapper `$()` può accettare come secondo argomento un oggetto letterale contenente gli attributi e i metodi dell'elemento creato nel primo argomento. In realtà avremmo anche potuto usare il metodo `html()` all'interno di tale oggetto letterale.

Potete visionare l'esempio finale in questa pagina.

jQuery: l'oggetto Event

jQuery normalizza l'oggetto `event` seguendo gli standard del W3C. L'oggetto in questione viene quindi passato al gestore di eventi e molte sue proprietà vengono normalizzate per poter essere usate in modo cross-browser.

Il costruttore jQuery.Event

Questo costruttore viene reso accessibile e può essere usato in combinazione con trigger(). Esempio:

```
// Crea un nuovo oggetto Event senza l'operatore new var e = jQuery.Event("click");  // Lancia un evento click fittizio jQuery("body").trigger( e );
```

A partire da jQuery 1.6 si può anche passare un oggetto al costruttore `jQuery.Event()` e le sue proprietà verranno impostate sul nuovo oggetto `Event`.

Esempio:

```
// Crea un nuovo oggetto Event con le proprietà dell'evento var e = jQuery.Event("keydown", { keyCode: 64 });  // Lancia un evento keydown fittizio con keyCode 64 jQuery("body").trigger( e );
```

Proprietà di Event

Le seguenti proprietà fanno parte dell'oggetto `Event`, ma alcuni dei loro valori possono essere `undefined` a seconda dell'evento:

altKey, attrChange, attrName, bubbles, button, cancelable, charCode, clientX, clientY, ctrlKey, currentTarget, data, detail, eventPhase, fromElement, handler, keyCode, layerX, layerY, metaKey, newValue, offsetX, offsetY, originalTarget, pageX, pageY, prevValue, relatedNode, relatedTarget, screenX, screenY, shiftKey, srcElement, target, toElement, view, wheelDelta, which

jQuery normalizza le seguenti proprietà per essere usate in modo cross-browser:

- `target`
- `relatedTarget`
- `pageX`
- `pageY`
- `which`
- `metaKey`

Proprietà speciali

Alcuni eventi nativi possono avere proprietà speciali che possono essere usate come proprietà dell'oggetto `event.originalEvent`. Per redere queste proprietà disponibili in tutti gli oggetti `Event`, possiamo aggiungerle all'array `jQuery.event.props`.

Esempio:

```
// Aggiunge la proprietà dataTransfer per essere usata con l'evento 'drop' nativo // per ottenere informazioni sui file trascinati nella finestra del browser
jQuery.event.props.push("dataTransfer");
```

jQuery: l'oggetto Deferred

`jQuery.Deferred()`, introdotto in jQuery 1.5, è un oggetto di utility che può registrare funzioni di callback multiple e memorizzarle in una coda, invocare una o più code, e gestire lo stato di successo o di fallimento di qualsiasi funzione sincrona o asincrona.

Come funziona l'oggetto Deferred

In JavaScript è prassi comune invocare funzioni che possono accettare dei callback invocati in quella funzione. Per esempio, nelle versioni di jQuery precedenti alla 1.5, i processi asincroni come `jQuery.ajax()` accettano callback da invocare in caso che la richiesta AJAX abbia avuto successo o meno o sia stata completata.

`jQuery.Deferred()` introduce diversi miglioramenti nel modo con cui i callback sono gestiti ed invocati. In particolare, `jQuery.Deferred()` fornisce dei modi flessibili per creare callback multipli, e questi callback possono essere invocati a prescindere dal fatto che il callback originale ha già avuto luogo. Questo oggetto è basato sul design di CommonJS Promises / A.

Bisogna pensare a questo oggetto come una funzione wrapper che genera una catena. I metodi deferred.then(), deferred.done() e deferred.fail() specificano le funzioni da chiamare e deferred.resolve(args) o deferred.reject(args) chiamano le funzioni con gli argomenti che voi gli fornite. Una volta che un oggetto `Deferred` è stato risolto o rifiutato permane in quello stato; una seconda chiamata al metodo `resolve()` viene quindi ignorata. Se più funzioni vengono aggiunte tramite il metodo `then()` dopo che l'oggetto `Deferred` è stato risolto, esse vengono eseguite immediatamente con gli argomenti forniti in precedenza.

Il costruttore jQuery.Deferred()

Questo costruttore crea un nuovo oggetto `Deferred` (l'operatore `new` è facoltativo). A questo oggetto può essere passata una funzione che viene eseguita prima della creazione dell'oggetto restituito dal costruttore. Questa funzione riceve come argomento l'oggetto `Deferred` e l'oggetto `this`. La funzione può usare diversi callback usando ad esempio il metodo `then()`.

L'oggetto parte in uno stato di tipo unresolved (non risolto). Qualsiasi callback aggiunto con i metodi `then()`,`done()` o `fail()` viene messo in coda per essere eseguito dopo.

Usare `resolve()` o deferred.resolveWith() fanno passare l'oggetto alla stato di risolto e quindi le funzioni impostate tramite `done()` vengono eseguite immediatamente. Chiamare invece `reject()` o deferred.rejectWith() fanno passare l'oggetto allo stato di respinto (rejected) e quindi le funzioni impostate tramite `fail()` vengono eseguite immediatamente. Una volta che l'oggetto è passato ad uno di questi stati, rimane in tale stato. Le funzioni di callback che vengono quindi aggiunte all'oggetto vengono eseguite subito.

I metodi di questo oggetto possono essere concatenati come qualsiasi altro oggetto di jQuery. Dopo la sua creazione, si possono usare i metodi elencati di seguito sia concatenandoli direttamente all'oggetto che salvando l'oggetto in una variabile e invocando i metodi su quella variabile.

- deferred.always()
- deferred.done()
- deferred.fail()
- deferred.isRejected()
- deferred.isResolved()
- deferred.pipe()
- deferred.promise()
- deferred.reject()
- deferred.rejectWith()
- deferred.resolve()
- deferred.resolveWith()
- deferred.then()
- .promise()

jQuery: CSS e animazioni

In questo articolo vorrei affrontare il discorso inerente il rapporto tra le proprietà e i valori CSS e le animazioni jQuery ottenibili tramite il metodo `animate()`. Questo rapporto deve essere chiaro se si vogliono creare animazioni di una certa complessità.

Step e valori CSS

Un'animazione jQuery si basa sul rapporto asimmetrico esistente tra il numero di fasi (step) di un'animazione e la durata dell'animazione stessa. Determinante in questo rapporto è il valore di easing scelto, che per impostazione predefinita è `swing`. Un'animazione jQuery modifica un valore iniziale di una proprietà CSS fino a portarlo al valore finale specificato dall'animazione. Per esempio:

```
$('#test').animate({   left: 100 }, 800);
```

Supponendo che il valore iniziale della proprietà CSS `left` sia 0, possiamo renderci conto del rapporto tra step e valori CSS utilizzando il metodo `step()`:

```
var step = 0;  $('#test').animate({   left: 100 }, {         duration: 800,         step:
function() {                  step++;                              console.log(step + ' '
+ $(this).css('left'));               }  });
```

Un ipotetico output sarebbe il seguente:

Step	Valore CSS
1	2px
2	3px
3	3px
4	4px

e così via fino a raggiungere il valore di 100 pixel. La progressione di valori è influenzata dal tipo di easing scelto: se avessimo scelto un tipo di easing `linear` i valori sarebbero stati diversi.

Proprietà CSS supportate da animate()

Il metodo `animate()` supporta le seguenti proprietà CSS:

1. Proprietà dei margini
2. Proprietà del padding
3. Larghezza a altezza
4. Interlinea
5. Dimensione dei font
6. Posizionamento
7. Visibilità
8. Opacità

Alcuni valori non standard comprendono:

1. `hide`
2. `show`
3. `toggle`

Per tutte le altre proprietà il supporto si può ottenere tramite plugin, come nel caso dei colori del testo e dello sfondo di un elemento.

Unità di misura

Un valore senza unità di misura viene considerato come espresso in pixel. Unità di misura come em, percentuali e pixel sono ugualmente supportate. Inoltre `animate()` supporta gli operatori di aggiunta e sottrazione di valori, che vanno usati sempre tra virgolette, ossia:

1. '+=100px'
2. '-=100px'

In questo caso al valore corrente della proprietà viene aggiunto o sottratto il valore specificato da questi operatori. Esempio:

```
$('#test').animate({   left: 100 }, 800, function() {        $(this).animate({
        left: '-=100px'        }, 800);  });
```

Specificità degli stili

Le animazioni modificano gli stili degli elementi usando l'oggetto `style` di ciascun elemento. Quindi il nostro primo esempio ha effettivamente questa struttura nel DOM:

[xml] ... [/xml]

Ciò significa che questi stili hanno una maggiore specificità rispetto agli stili normali del nostro CSS. Per resettare questi stili possiamo sia usare il metodo `css()`, che opera anch'esso sull'oggetto `style`, o usare gli stili del nostro CSS facendo in modo che abbiano una specificità maggiore. Questo si può ottenere sia con la cascata che con la dichiarazione `!important`.

jQuery: il metodo prop()

jQuery 1.6 introduce il metodo `prop()` per gli elementi del set di jQuery. Questo metodo accetta come argomento il nome della proprietà da leggere (getter). Può anche essere usato per impostare una proprietà, nel qual caso può anche accettare un oggetto letterale contenente i nomi e i valori delle proprietà da impostare (setter). Vediamolo in dettaglio.

prop() come getter

Il metodo `.prop()` restituisce il valore della proprietà solo per il *primo* elemento nel set di jQuery. Restituisce `undefined` se il valore della proprietà non è stato impostato o se il set non ha elementi. Per ottenere il valore di ciascun elemento si può usare un ciclo `.each()` o il metodo `.map()`.

La documentazione ufficiale di jQuery spiega il perchè dell'introduzione di questo metodo spiegando la differenza tra attributi e proprietà:

The difference between *attributes* and *properties* can be important in specific situations. **Before jQuery 1.6**, the `.attr()` method sometimes took property values into account when retrieving some attributes, which could cause inconsistent behavior. **As of jQuery 1.6**, the `.prop()` method provides a way to explicitly retrieve property values, while `.attr()` retrieves attributes.

Ora quindi `.prop()` gestisce le proprietà e `.attr()` gli attributi. Per esempio, considerando il caso di una checkbox con l'attributo `checked` impostato avremo:

`elem.checked`	`true` (Booleano) Cambia con lo stato della checkbox
`$(elem).prop("checked")`	`true` (Booleano) Cambia con lo stato della checkbox
`elem.getAttribute("checked")`	`"checked"` (Stringa) Stato iniziale della checkbox che non cambia
`$(elem).attr("checked")`*(1.6)*	`"checked"` (Stringa) Stato iniziale della checkbox che non cambia
`$(elem).attr("checked")`*(1.6.1+)*	`"checked"` (Stringa) Cambia con lo stato della checkbox
`$(elem).attr("checked")`*(prima della 1.6)*	`true` (Booleano) Modificato con lo stato della checkbox

Secondo le specifiche dei form, l'attributo `checked` è un *attributo booleano*, il che significa che la proprietà corrispondente è `true` quando l'attributo è presente ed ha un valore. Il modo migliore per stabilire se una checkbox è selezionata basandosi sul valore della sua proprietà è uno dei seguenti:

- `if ( elem.checked )`
- `if ( $(elem).prop("checked") )`
- `if ( $(elem).is(":checked") )`

La documentazione di jQuery precisa:

If using jQuery 1.6, the code `if ( $(elem).attr("checked") )` will retrieve the actual content *attribute*, which does not change as the checkbox is checked and unchecked. It is meant only to store the default or initial value of the checked property. To maintain backwards compatability, the `.attr()` method in jQuery 1.6.1+ will retrieve and update the property for you so no code for boolean attributes is required to be changed to `.prop()`. Nevertheless, the preferred way to retrieve a checked value is with one of the options listed above.

Ossia il metodo consigliato per questo tipo di proprietà (`checked`) è `.prop()`.

prop() come setter

Il metodo `.prop()` può anche essere usato per impostare una o più proprietà di un elemento. I modi sono i seguenti:

Singola proprietà

```
$('input').prop('checked', true);
```

Molteplici proprietà

```
$('input').prop({      disabled: false,       checked: true });
```

Valore calcolato tramite funzione di callback

```
$("input[type='checkbox']").prop('checked', function( i, val ) {   return !val; });
```

`val`, il valore della proprietà dell'elemento, viene in questo caso impostato sul valore booleano opposto.

jQuery: risoluzione dei problemi in Internet Explorer 6 e 7

Anche se, tecnicamente parlando, è sempre jQuery che supporta un browser e non il contrario, c'è da dire che i risultati che si ottengono in Internet Explorer 6 e 7 sono di gran lunga inferiori a quelli ottenibili in altri browser. Ho calcolato che su 200 test da me eseguiti con jQuery, IE 6 e 7 ne superavano complessivamente meno del 60%. Questa carenza di supporto è data dal fatto che si tratta

di browser ampiamente obsoleti e su cui la stessa Microsoft preferisce stendere un velo pietoso e puntare sulle versioni 9 e 10 di Explorer. Tuttavia, statistiche alla mano, sono browser ancora diffusi. In alcuni paesi la versione 6 è addirittura ben al di sopra del 5%, ossia la fatidica soglia al di sotto della quale un browser può anche essere ignorato. In questo articolo vorrei illustrare alcune tecniche e accorgimenti per risolvere i problemi più noti con questi due browser.

Animazioni

IE 6 e 7 hanno dei notevoli problemi con le animazioni complesse ed annidate. Nello specifico, i problemi riguardano soprattutto il metodo `animate()`, ossia:

1. le parole chiave `hide`, `show`, `toggle`
2. la velocità d'esecuzione delle animazioni (intervalli superiori ai 2000 millisecondi)

Si possono risolvere questi problemi nel modo seguente:

1. sostituire `animate()` con i metodi specifici `fadeIn()`, `fadeOut()`, `hide()` e `show()`
2. evitare di usare intervalli superiori ai 2000 millisecondi

Non ho riscontrato significativi problemi nel metodo `animate()` con le seguenti proprietà CSS:

1. `top`, `right`, `bottom`, `left`
2. `width`, `height`
3. `opacity` (a parte sulle immagini PNG - vedi articolo)
4. padding e margini
5. `font-size`
6. `line-height`

Una speciale attenzione meritano gli operatori `+=` e `-=` usati con i valori delle proprietà CSS. A volte sia IE 6 che 7 restituiscono un errore, sostenendo di non essere in grado di leggere il valore della proprietà specificata. Si risolve in questo modo:

```
var ratio = 100;   $(element).click(function() {      var left = $(this).position().left;
        $(this).animate({              left: left + ratio     }, 'slow');  });
```

invece che:

```
$(element).click(function() {                          $(this).animate({              left:
'+=100px'       }, 'slow');  });
```

Gestione degli errori

In queste versioni di Internet Explorer il codice va scritto passo passo. Quando si affrontano costrutti complessi, è bene usare la gestione delle eccezioni offerta da JavaScript:

```
if(ie) {        try {                  // codice complesso            }  catch(e) {
```

Consiglio caldamente di usare degli strumenti di sviluppo specifici per Internet Explorer 6 e 7, come ad esempio i seguenti. Usate spesso la console DOM e JavaScript dopo ogni azione eseguita dal vostro codice per verificare che IE l'abbia interpretata correttamente.

Bug CSS

I bug di IE 6 e 7 con i CSS sono quasi tutti collegati alla proprietà hasLayout. Per questo motivo quando si creano delle animazioni occorre innanzitutto testare prima gli stili CSS per essere sicuri che non ci siano problemi. Se il vostro codice modifica gli stili di un elemento, create prima una classe CSS ed applicatela all'elemento per osservare quale sarà la resa finale in Explorer.

Performance

Questi browser hanno una performance molto scarsa. Per questo motivo:

1. usate `innerHTML` invece che `html()`
2. usate i cicli `for` e `for...in` invece che `.each()` e `$.each()`
3. assemblate il contenuto HTML come array e infine trasformatelo in stringa

Spieghiamo l'ultimo punto:

```
var html = [], i = 0; html[i] = '<ul>';  for(var j = 1; j < 10000; j += 1) {
        html[i++] = '<li>' + j + '</li>';  }  html[html.length+1] = '</ul>';  var str =
html.join('');  $(element)[0].innerHTML = str;
```

Questo approccio migliora l'esecuzione di circa il 30/40% in Internet Explorer 6 e 7.

Downgrade di jQuery

A volte usare l'ultima versione di jQuery può creare problemi in IE 6 e 7. Anche se sulla carta jQuery dovrebbe funzionare su questi browser, molto spesso si verificano alcuni comportamenti imprevisti relativi alle modifiche apportate al codice della libreria nelle nuove versioni. Per questo motivo è spesso decisivo effettuare un downgrade di jQuery alla versione che si rivela più stabile in questi browser.

Per esperienza posso dirvi che la versione 1.4 si è rivelata funzionare meglio delle ultime versioni. Quindi potete dare ad IE 6 e 7 il seguente codice:

[xml] <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4/jquery.min.js"></script> [/xml]

Vi consiglio di utilizzare uno script lato server per dare solo alle versioni di Explorer selezionate questa versione specifica di jQuery.

Individuare Internet Explorer

A mio avviso il modo migliore per dare del codice solo ad alcune versioni di Internet Explorer inferiori alla 10 sono ancora i commenti condizionali. Tuttavia, esistono anche altri metodi per farlo usando jQuery, e sono riassunti in questa discussione su Stack Overflow.

Un metodo che potrebbe soppiantare l'uso dei commenti condizionali è quello dell'individuazione delle caratteristiche supportate da Internet Explorer. Il problema di questo metodo è che è meno rapido e pratico dei commenti condizionali.

jQuery: animazioni e timer

I timer JavaScript in jQuery servono per creare animazioni cicliche e ripetute nel tempo. Il segreto del loro funzionamento sta nell'intervallo (in millisecondi) specificato per ogni ripetizione del timer. Esso deve sempre essere uguale alla somma degli intervalli delle animazioni. Vediamo due esempi.

Nel primo esempio, un elemento appare e scompare modificando la sua opacità:

```
setTimeout(function() {              $('#test').animate({opacity: 'toggle'}, 2000);
setTimeout(arguments.callee, 2000);          }, 500);
```

La seconda chiamata a `setTimeout()` crea un timer ciclico, in quanto `arguments.callee` è un riferimento alla funzione del primo timer più esterno. Notate come l'intervallo del secondo timer è uguale alla durata dell'animazione (2000 millisecondi).

Il secondo esempio, invece, un elemento si sposta da destra a sinistra (e viceversa):

```
setTimeout(function() {                  $('#test2').animate({left: 150}, 1000, function()
{            $(this).animate({left: 0}, 1000);                    });
setTimeout(arguments.callee, 2000);                            }, 500);
```

Anche in questo caso la durata del secondo timer è uguale alla durata complessiva delle due animazioni (1000 + 1000 millisecondi).

setTimeout() o setInterval()?

Per quanto riguarda il controllo sulle animazioni jQuery, `setTimeout()` è da preferire in quanto con questo metodo riusciamo a gestire anche il tempo di inizio dell'animazione. Con `setInterval()` questo non è possibile, in quanto questo metodo accetta un solo valore temporale.

Potete visionare l'esempio finale in questa pagina.

jQuery e il DOM: sguardo comparativo

Gran parte del lavoro svolto da jQuery avviene sul DOM (Document Object Model). Semplificando al massimo la sua definizione, il DOM è una rappresentazione ad albero di un documento strutturato (HTML, XHTML, XML, eccetera). Questo albero è rovesciato, ossia parte dalla radice e si sviluppa in rami e nodi. In questo articolo vedremo alcuni metodi e selettori di jQuery comparandoli con il loro equivalente nelle specifiche DOM.

ID e classi

	ID	Classi
jQuery	`$('#test')`	`$('.test')`
DOM	`document.getElementById('test')`	`document.getElementsByClassName('test')`

Il secondo metodo DOM non è implementato in tutte le versioni di Internet Explorer. Ecco una soluzione:

```
if(typeof document.getElementsByClassName !== 'function') {  function
getElementsByClassName(className, tag, elm){  var testClass = new RegExp("(^|\\s)" +
className + "(\\s|$)");        var tag = tag || "*";  var elm = elm || document;     var
elements = (tag == "*" && elm.all)? elm.all : elm.getElementsByTagName(tag);        var
returnElements = [];   var current;   var length = elements.length;  for(var i=0; i<length;
i++){          current = elements[i];                if(testClass.test(current.className)){
                       returnElements.push(current);         }       }       return
returnElements; }  }
```

Primo e ultimo elemento

	Primo elemento	Ultimo elemento
jQuery	`$('div:first')`	`$('div:last')`
DOM	`element.firstChild`	`element.lastChild`

Quando si usano le proprietà DOM `firstChild` e `lastChild` occorre sempre verificare che il nodo selezionato sia un elemento, ossia che la sua proprietà `nodeType` abbia valore 1:

```
var element = document.getElementById('test'); var first = null;
if(element.firstChild.nodeType == 1) {        first = element.firstChild;  }
```

Aggiungere e rimuovere classi CSS

	Aggiungere una classe CSS	Rimuovere una classe CSS
jQuery	`$(element).addClass('test')`	`$(element).removeClass('test')`
DOM	`element.className`	`element.className`

In teoria sarebbe solo necessario usare i metodi DOM `setAttribute()` e `removeAttribute()`, ma questi ultimi non sono ben supportati nelle versioni di Internet Explorer inferiori alla 8. Quindi:

```
function addClass(element, className) {       element = document.getElementById(element);
        element.className += className;  }  function removeClass(element, className) {
        element = document.getElementById(element);   var replaced = new RegExp("(^|\\s)" +
className + "(\\s|$)");        var klass = element.className;
        element.className = klass.replace(replaced, '');        }
```

Elemento precedente e successivo

	Elemento precedente	Elemento successivo
jQuery	`$(element).prev()`	`$(element).next()`
DOM	`element.previousSibling`	`element.nextSibling`

Quando un nodo non esiste nel DOM, il suo valore è `null`. Quindi per usare le proprietà `previousSibling` e `nextSibling` dobbiamo appunto eseguire questa verifica:

```
var element = document.getElementById('test'); var next = null;  if(element.nextSibling !==
null) {         next = element.nextSibling;  }
```

Figli di un elemento

	Figli di un elemento
jQuery	`$(element).children()`
DOM	`element.childNodes`

La proprietà `childNodes` è una NodeList che contiene tutti i nodi figli di un elemento. Questo tipo di struttura DOM non è un array ma possiede una proprietà `length` e si può accedere ai suoi membri tramite la consueta notazione degli array:

```
function children(element) {   element = document.getElementById(element);   var elements =
[],     nodes = element.childNodes,    len = nodes.length,    i;              for(i = 0; i <
len; i += 1) {                 var node = nodes[i];
        if(node.nodeType == 1) {                                    elements.push(node);
```

Riferimenti

1. JavaScript and HTML DOM reference
2. DOM (Document Object Model) Reference

jQuery: le proprietà selector e context

Ogni espressione jQuery ha due importanti proprietà, `selector` e `context`. La prima restituisce una stringa contenente i selettori CSS usati nell'espressione, mentre la seconda restituisce il contesto in cui l'espressione ha luogo, partendo dall'oggetto globale del DOM `HTMLDocument`. Vediamone insieme i dettagli.

Partiamo da questa struttura:

[xml] ... ... [/xml]

Una semplice espressione jQuery è la seguente:

```
var $p = $('p', '#test');
```

che ha i seguenti valori nelle proprietà `selector` e `context`:

```
alert($p.selector); // #test p alert($p.context);  // [object HTMLDocument]
```

La proprietà `selector` restituisce ciò che ci aspettavamo, ma `context` usa l'intero oggetto `document`. Per avere risultati più coerenti possiamo passare un'espressione DOM come secondo parametro dell'espressione jQuery:

```
var $p2 = $('#test p', document.body); alert($p2.context); // [object HTMLBodyElement]
```

Come si può notare, ora il contesto è cambiato. Volendo essere ancora più specifici avremmo potuto usare l'espressione DOM `document.getElementById('test')`. Ricordate che le espressioni DOM sono notevolmente più performanti di quelle jQuery.

Primi passi con gli oggetti JavaScript per sviluppatori jQuery

Tempo fa Davide mi ha fatto notare che molti miei articoli sulla programmazione orientata agli oggetti erano interessanti ma piuttosto ostici per chi non mastica pane e programmazione. In questo articolo vorrei affrontare nuovamente l'argomento dal punto di vista di chi usa jQuery ma non è molto addentro alle problematiche relative agli oggetti.

Oggetti: cosa sono?

Quando descriviamo un oggetto reale parliamo del suo colore, della sua forma, del suo peso e così via, ossia parliamo delle sue proprietà. Le proprietà non sono altro che dei tratti distintivi dell'oggetto. Queste proprietà possono avere dei valori, ad esempio:

```
Scatola:        colore = blu    forma = rettangolare    peso = 5Kg
```

Una scatola è un oggetto, e come potete vedere ha proprietà e valori. Gli oggetti JavaScript sono la stessa cosa: una collezione di coppie di proprietà e valori.

Tuttavia gli oggetti sarebbero alquanto inutili se contenessero solo dati. In fondo, penserete, ci sono già gli array per questo. Invece gli oggetti possono anche compiere delle azioni su questi dati, ossia sulle loro proprietà. Queste azioni si chiamano metodi, e sono le tradizionali funzioni JavaScript che già conoscete:

```
Scatola:        colore = blu    forma = rettangolare    peso = 5Kg      mostraPeso = funzione()
{                       alert('Il peso è di ' + peso);                  }
```

Come si scrivono gli oggetti in JavaScript

Fino ad ora abbiamo usato una pseudo-rappresentazione degli oggetti. Vediamo ora come vengono effettivamente scritti in JavaScript.

Oggetti letterali

Questi oggetti li avrete sicuramente usati nelle opzioni dei plugin jQuery. Hanno questa struttura:

```
var Oggetto = {    proprieta: valore,    proprieta2: valore,    //...        proprietaN:
valore };
```

Ciascuna coppia proprietà/valore ha come separatore una virgola, tranne l'ultima. Le proprietà vengono separate dai loro valori tramite i due punti. Vediamo la rappresentazione del nostro oggetto `Scatola`:

```
var Scatola = {        colore: 'blu',         forma: 'rettangolare',                peso:
'5Kg',          mostraPeso: function() {                      alert(this.peso);
        }   };
```

Non preoccupatevi per ora della parola chiave `this`. Concentriamoci invece su come i dati vengono scritti. Noterete infatti che i primi tre valori sono delle stringhe e quindi vanno racchiusi tra virgolette. Negli oggetti letterali infatti bisogna osservare strettamente le regole sui tipi di dati in JavaScript. Quindi i numeri, i riferimenti ad altri oggetti, i booleani, le espressioni regolari, `null`, `undefined` e gli array non hanno bisogno delle virgolette, ma le stringhe si.

Oggetti con costruttori

Per capire questi oggetti pensiamo a come viene realizzata una scatola. Prima si fa uno stampo della scatola e poi si crea fisicamente la scatola in base a quello stampo. In altre parole, prima si crea il modello e poi l'oggetto concreto.

Ecco come viene creato il modello con questi oggetti (lo stampo della scatola):

```
function Scatola(colore, forma, peso) {       this.colore = colore;  this.forma = forma;
        this.peso = peso;   }
```

La funzione `Scatola` accetta come suoi parametri i tre valori che andranno a definire le sue proprietà, ossia `colore`, `forma` e `peso`. Il nostro stampo quindi ci dice che l'oggetto che andremo a creare dovrà avere queste proprietà e come valori i valori passati alla funzione.

Ma come ci crea la scatola reale? Tramite l'operatore `new` posto davanti a `Scatola`:

```
var scatola = new Scatola('blu', 'rettangolare', '5Kg');
```

Il nostro oggetto finale, la nostra scatola stampata, è ora tutto in `scatola`, che contiene tutte le sue proprietà e i suoi valori.

Riassumendo: gli oggetti letterali creano in un unico passaggio il nostro oggetto finito, mentre gli oggetti con costruttori creano prima un modello dell'oggetto e poi l'oggetto finito.

Cos'è this?

Avrete capito che `this` è una parola chiave JavaScript. Nell'àmbito degli oggetti, `this` è un riferimento all'oggetto corrente. In altre parole, nel nostro caso è un alias di `Scatola`. Quindi nell'esempio degli oggetti con costruttore avremmo potuto anche scrivere:

```
function Scatola(colore, forma, peso) {       Scatola.colore = colore;       Scatola.forma =
forma;  Scatola.peso = peso;   }
```

E nell'esempio dell'oggetto letterale:

```
var Scatola = {        colore: 'blu',         forma: 'rettangolare',                peso:
'5Kg',         mostraPeso: function() {                      alert(Scatola.peso);
        }   };
```

Attenzione a quando si usa questa parola chiave con jQuery. Bisogna sempre chiedersi: `this` si riferisce all'oggetto che penso? Non sempre. Infatti jQuery crea a sua volta degli oggetti e se usiamo questa parola chiave all'interno dei suoi oggetti o metodi quest'ultima farà riferimento a jQuery e non all'oggetto che vogliamo.

Esempio:

```
var Animator = {       element: '#test',      fade: function() {
        $(this.element).fadeOut('slow');              }  };
```

Qui `this` si trova all'interno di `$()` ossia dell'oggetto `jQuery` e quindi fa riferimento a `jQuery` e non ad `Animator`! Il risultato restituirà `undefined` (un errore), perchè `element` non esiste come proprietà di `jQuery`.

Accedere alle proprietà di un oggetto

Si può accedere alle proprietà di un oggetto in due modi:

1. con la notazione puntata (`oggetto.proprieta`)
2. con la notazione a parentesi quadre (`oggetto['proprieta']`)

Esempi:

```
alert(Scatola.peso); // '5Kg' alert(Scatola['peso']); // '5Kg'
```

Nel caso degli oggetti con costruttori bisogna prima usare l'operatore `new`:

```
var scatola = new Scatola('blu', 'rettangolare', '5Kg'); alert(scatola.peso); // '5Kg'
alert(scatola['peso']); // '5Kg'
```

Se la proprietà è una funzione, ossia un metodo dell'oggetto, occorre usare anche le doppie parentesi tonde, come in qualsiasi altra invocazione di funzione:

```
Scatola.mostraPeso(); // '5Kg' Scatola['mostraPeso']();  // '5Kg'
```

Linee guida per la creazione di plugin jQuery

La percentuale di plugin jQuery realmente utile è straordinariamente bassa. Volendo stimare il loro numero, dobbiamo purtroppo dire che i plugin usati più di una volta nel nostro lavoro si contano sulle dita di una mano. Nel mio caso sono Cycle, FancyBox e Colorbox. Tre. La mia opinione è però condizionata dal fatto che in genere mi piace più risolvere i problemi in modo autonomo che affidarmi ai plugin. L'unica eccezione sono gli slideshow con effetti particolari. Per quelli in genere uso plugin, specie quando le deadline non mi consentono di trovare una soluzione personalizzata. Anche in quel caso ho però notato una certa omologazione nel modo di scrivere plugin: infatti, svolgono quasi tutti gli stessi compiti con gli stessi effetti. Ma l'aspetto più negativo da me notato è la scarsa usabilità, flessibilità e adattabilità della stragrande maggioranza dei plugin. In questo articolo vorrei dare delle indicazioni per scrivere plugin che effettivamente risolvano dei problemi piuttosto che crearne di nuovi.

Specificità

Sforzatevi di scrivere plugin che svolgano un solo compito specifico ma lo svolgano bene. Chiedetevi sempre: qual'è un problema ricorrente nell'implementazione front-end di un sito che necessiterebbe di un plugin jQuery? Esempio: clear automatico degli elementi flottati senza dover aggiungere manualmente classi CSS.

Soluzione:

```
(function($) {          $.fn.clearFloats = function(options) {                          var
that = this;            options = $.extend({                    mode: 'overflow'
        }, options);                            return that.each(function() {
                                                                  case 'overflow':
                                                parent.height(that.height());
                                                break;
        default:                                                  parent.css('overflow',
'hidden');                                                      break;
                                });              };     })(jQuery);
```

Un solo compito, ma svolto nel migliore dei modi possibili. Evitate di scrivere plugin che vogliano fare tutto il possibile e l'immaginabile. Concentratevi su quello di cui c'è veramente bisogno.

Usabilità

I vostri plugin dovrebbero essere semplici da usare. Se per capire come funziona il vostro plugin ci vogliono ore di tentativi, allora sicuramente c'è qualcosa che non funziona a livello di usabilità.

Per prima cosa, usate il giusto numero di opzioni. Ricordate che potete sempre creare in futuro nuove versioni del vostro plugin, quindi non c'è bisogno che esauriate da subito tutte le opzioni possibili. Non esagerate con le opzioni: un utente in genere ne userà veramente poche, ossia quelle che gli servono per far funzionare bene il plugin, ma nulla più.

Un altro aspetto fondamentale sono la documentazione e gli esempi: dovete fornire una documentazione dettagliata, descrivendo tutte le opzioni e gli usi del vostro plugin. Inoltre dovete fornire degli esempi "live", liberamente consultabili.

Includete nel file del download tutti gli esempi e la documentazione, più tutte le dipendenze necessarie (immagini, CSS, eccetera). Includete anche una copia di jQuery, in modo che l'utente possa usare gli esempi offline. Infine, ricordatevi di includere una copia minificata del vostro plugin nel download (chiamatela `jquery.plugin.min.js`).

Metadati nel codice

Commentate accuratamente il vostro codice, specie per quegli utenti avanzati che potrebbero voler modificare il vostro plugin. Inserite dei metadati all'inizio del plugin, specificando una descrizione del plugin, la versione di jQuery richiesta (fondamentale) e la licenza d'uso:

```
/**   * Plugin: Descrizione   *   * @author Gabriele Romanato   * @version 1.0   * @require jQuery 1.5+   * @license GPL                                 */
```

Nel codice, cercate di spiegare quei passaggi chiave fondamentali per il funzionamento del plugin. Esempio:

```
var parent = that.parent(); // Il clear va applicato al diretto genitore del float
```

L'ideale sarebbe usare JSDoc, ma anche dei commenti semplici ma efficaci possono servire allo scopo.

Consigli per avere selettori jQuery più efficienti

Alcuni giorni fa *Craig Buckler* ha pubblicato un interessante articolo su Sitepoint in cui presentava cinque modi per aumentare la performance dei selettori jQuery. In questo articolo vorrei riprendere il discorso di Craig aggiungendovi delle precisazioni che potranno esservi utili nel vostro lavoro con jQuery.

Usare un ID se possibile

Un selettore di ID è di gran lunga il metodo più veloce per selezionare un elemento con jQuery:

```
$('#test')
```

Tuttavia, ciò non è ancora il massimo. Infatti se usiamo la sintassi CSS il selector engine di jQuery deve comunque operare una conversione. Se il browser supporta il metodo `querySelector()`, verrà usato questo metodo, altrimenti il metodo tradizionale`getElementById()`.

Quindi il sistema più veloce prevede appunto l'uso dell'ultimo metodo DOM citato:

```
$(document.getElementById('test'))
```

Usare i selettori di classe in modo specifico

Nei CSS esiste una distinzione tra `.test` e `div.test`. Il primo è un selettore di classe generico (applicabile a tutti gli elementi), mentre il secondo è un selettore di classe di un elemento specifico. In jQuery il secondo è molto più performante del primo:

```
$('div.test')
```

Nei browser che supportano il DOM in modo completo, potete usare il metodo `getElementsByClassName()`, che è in assoluto il metodo più veloce per gestire le classi. Perchè? Perchè i metodi del DOM vengono implementati con linguaggi come C/C++, di gran lunga molto più veloci delle soluzioni JavaScript:

```
$(document.getElementsByClassName('test'))
```

Semplificare i selettori

Bisognerebbe evitare selettori complessi come:

```
$('body #page:first-child article.main p#intro em')
```

e invece usare selettori come:

```
$('p#intro em')
```

Tuttavia esiste un altro modo per semplificare i selettori ed è quello di usare il loro *contesto*. Il contesto infatti viene passato come secondo parametro al wrapper `$()` e specifica il nodo o il punto del DOM in cui jQuery deve cominciare a cercare. Per impostazione predefinita il contesto è sempre `document`.

Quindi avremo:

```
$('em', document.getElementById('intro'))
```

che è molto più performante.

Evitare la ripetizione dei selettori

Bisognerebbe sempre evitare di ripetere i selettori, come in questo caso:

```
$('p').css('color', 'green'); $('p').text('Test');
```

e usare invece la concatenazione di jQuery:

```
$('p').css('color', 'green').text('Test');
```

oppure usare una copia cache del selettore:

```
var $p = $('p'); $p.css('color', 'green'); $p.text('Test');
```

In tal senso Craig ci avverte che:

A differenza delle collezioni di nodi DOM, le collezioni jQuery non sono live e l'oggetto non viene aggiornato quando gli elementi vengono aggiunti o rimossi dal documento. Potete evitare questo problema creando una collezione DOM e passandola al wrapper `$()` quando richiesto.

Quindi avremo:

```
var p = document.getElementsByTagName('p'); $(p).css('color', 'green'); // aggiornamento
del DOM $(p).text('Test');
```

jQuery: i filtri not() e filter()

I filtri jQuery `not()` e `filter()` hanno identico scopo ma funzionalità e modalità diverse. Entrambi infatti escludono uno o più elementi dal set restituito dalla selezione di jQuery, ma il primo utilizza i selettori mentre il secondo sfrutta una funzione di callback. Vediamone insieme i dettagli.

Partiamo dalla seguente struttura:

[xml] <ul id="test"> <li class="a1">A <li class="b">B <li class="c">C <li class="d1">D E F G [/xml]

Definiamo per prima cosa una funzione helper per visualizzare i risultati nella console:

```
function print(element) {      var output = (element.attr('class')) ?
element[0].tagName.toLowerCase() + ' [' + element.attr('class') + ']' :
element[0].tagName.toLowerCase() + ' - ' + element[0].firstChild.nodeValue;
        console.log(output);  }
```

Diciamo che vogliamo selezionare solo gli elementi privi di una classe CSS. Possiamo in questo caso usare `not()`:

```
var ul = document.getElementById('test');  var $nots = $('li', ul).not('[class]');
$nots.each(function() {               print($(this)); // li - E, li - F, li - G      });
```

In pratica `not()` valuta i selettori passatigli come argomento e esclude dal nuovo set tutti quegli elementi che corrispondono a tali selettori. In questo caso usando un selettore di attributo abbiamo detto all'engine di jQuery di escludere quegli elementi aventi una classe CSS.

Ora vogliamo invece selezionare solo gli elementi la cui classe CSS termina con un numero. Qui possiamo usare `filter()`:

```
var $filtered = $('li', ul).filter(function() {
return /\d$/.test($(this).attr('class'));                                });
```

In pratica `filter()` valuta l'espressione restituita dalla funzione di callback (`return` è obbligatorio) e la compara con gli elementi del set per operare la selezione. Tutti quegli elementi che soddisfano l'espressione vengono inclusi nel nuovo set (e gli altri esclusi).

Altro esempio: vogliamo selezionare solo quegli elementi la cui classe CSS è composta da una sola lettera:

```
var $filtered2 = $('li', ul).filter(function() {
                                  }                  });
$filtered2.each(function() {          print($(this)); // li [b], li [c]      });
```

L'unico accorgimento preso in questo caso è quello di verificare che gli elementi abbiano effettivamente una classe CSS onde evitare valori `undefined`. Per il resto il procedimento è lo stesso di quello visto in precedenza.

jQuery: ordine di esecuzione degli eventi load e ready

Comprendere l'ordine di esecuzione degli eventi `load()` e `ready()` è di fondamentale importanza per poter caricare correttamente il nostro codice jQuery. In JavaScript, infatti, l'oggetto globale `window` viene creato prima dell'oggetto `document`. Da ciò consegue che l'ordine degli eventi viene determinato da questa sequenza. Possiamo sfruttare questa caratteristica per effettuare il precaricamento del codice. Vediamo come.

Il seguente codice jQuery **non** produrrà gli effetti sperati:

```
$(window).load(function() {    alert('Load');  });  $(function() {    alert('Ready');  });
```

Infatti vedremo solo il secondo alert, in quanto l'evento `ready()` sovrascriverà l'evento `load()`. Dobbiamo invece inserire l'evento `ready()` all'interno del contesto di esecuzione di `load()`:

```
$(window).load(function() {          alert('Load');          $(function() {
alert('Ready');               }); });
```

E questo è il risultato:

jQuery: differenza tra parent() e parents()

jQuery dispone di molti selettori per operare sul DOM. Il funzionamento di alcuni di questi selettori, tuttavia, non è molto chiaro agli sviluppatori per via della sintassi che li caratterizza. Tipico è il caso dei selettori `parent()` e `parents()`. In questo articolo vorrei enfatizzare le differenze esistenti tra questi due selettori.

parent()

Questo selettore seleziona l'immediato genitore di un elemento e dovrebbe essere usato quando la struttura del DOM presenta un solo genitore, come in questo caso:

[xml] Test [/xml]

Quindi possiamo scrivere:

```
var parent = $('#child').parent();
```

o in modo più specifico:

```
var parent = $('#child').parent('#parent');
```

Ma se la struttura fosse stata:

[xml]   Test   [/xml]

e avessimo scritto:

```
var grandParent = $('#child').parent('#grandparent');
```

non avremmo ottenuto l'effetto sperato, perchè in questo caso la struttura del DOM presenta due genitori (o antenati) e quindi va usato il selettore `parents()`.

parents()

Questo selettore seleziona uno dei genitori dell'elemento specificato e può essere usato quando la struttura del DOM si presenta con più genitori (o antenati), come in questo caso:

[xml]   Test   [/xml]

Quindi possiamo scrivere:

```
var grandParent = $('#child').parents('#grandparent');
```

Per funzionare correttamente, questo selettore ha bisogno sempre che voi specifichiate l'espressione CSS precisa con cui selezionare l'antenato (in questo caso è `#grandparent`).

jQuery: l'evento scroll()

Il mio amico Paolo Loschi mi ha recentemente sottoposto una serie di siti che utilizzano le nuove tecniche jQuery / CSS3 per la realizzazione di effetti in stile Flash. Molti dei siti proposti sono stati recensiti da CSS Winner e sono per lo più esempi di uso avanzato dello scrolling jQuery (verticale). In questo articolo vorrei parlare dell'evento `scroll()` di jQuery, poichè questo tipo di siti lo utilizza per associare azioni sui blocchi e le immagini.

Sintassi e descrizione

`scroll()` è una scorciatoia per `.bind('scroll', handler)` quando viene considerato come evento con relativa funzione di callback (handler) e per `.trigger('scroll')` quando viene usato direttamente per innescare l'azione sull'elemento (nessun parametro).

Questo evento viene legato ad un elemento solo quando l'utente scrolla l'elemento in questione (sia usando le scrollbar che la rotellina del mouse). In genere viene applicato all'oggetto globale `window`, ma può anche essere utilizzato su `iframe` o su quegli elementi in cui la proprietà CSS `overflow` è stata impostata su `scroll` (o anche su `auto`, qualora la larghezza o altezza dell'elemento è inferiore alla larghezza o altezza effettive dei suoi contenuti).

Esempi d'uso

L'evento può essere usato in questo modo:

```
$('#target').scroll(function() {      console.log('Scroll!'); });
```

L'azione associata all'evento viene eseguita ogni volta che l'utente esegue lo scrolling dell'elemento. L'evento può anche essere invocato manualmente, senza parametri:

```
$('#other').click(function() {   $('#target').scroll(); });
```

L'oggetto globale `window`, usato comunemente per gestire lo scrolling dell'intera finestra del browser, può anche essere utilizzato (come detto) sugli elementi `iframe` che, ricordiamolo, vengono considerati nel DOM come documenti a se stanti:

```
$(window).scroll(function() {         $('#navigation').css('top', $(window).scrollTop());
});
```

Il codice di cui sopra serve ad emulare il posizionamento fisso in Internet Explorer 6.

jQuery: il metodo clone()

Il metodo `clone()` di jQuery è un'implementazione cross-browser del metodo `cloneNode()` del DOM. Di fatto questo metodo clona un'intera struttura DOM che può essere in seguito riutilizzata in altre parti del documento. Vediamone insieme i dettagli.

Supponiamo di avere questa struttura:

[xml] <ul id="test"> A B C [/xml]

Per clonarla possiamo scrivere:

```
var copy = $('#test').clone();
```

Ora `copy` contiene l'intera struttura memorizzata come un oggetto jQuery. Per poterla riutilizzare dobbiamo usare il wrapper di jQuery:

```
$(copy).prependTo('#title');
```

Possiamo addirittura manipolare la nostra copia prima di reinserirla nel DOM:

```
$(copy).addClass('copy').prependTo('#title');
```

`clone()` accetta come parametro un valore booleano. `true` clona l'oggetto con tutti i suoi eventi, mentre `false` clona solo l'oggetto. Il valore predefinito è `false`.

jQuery: aspetti del DOM e della performance

jQuery semplifica di molto la gestione del DOM tramite JavaScript. Il DOM è una sigla che sta per *Document Object Model*, ossia modello a oggetti del documento. Un documento non è necessariamente un documento HTML, ma può essere scritto in qualsiasi linguaggio di marcatura supportato da un browser (XML, SVG, eccetera). Scopo di questo articolo è quello di chiarire alcuni aspetti legati al DOM e a jQuery.

Oggetti jQuery e oggetti DOM

jQuery, tramite il wrapper `$()`, trasforma un oggetto del DOM in un oggetto jQuery, assegnandogli i metodi e le proprietà della libreria. Si può operare un processo inverso riconvertendo l'oggetto jQuery in un oggetto del DOM. Per esempio, data la seguente marcatura:

[xml] [/xml]

Quello che segue è un oggetto jQuery:

```
var test = $('#test');
```

Possiamo quindi usare i metodi di jQuery:

```
console.log(test.attr('class')); // 'test'
```

Ma non quelli del DOM:

```
console.log(test.className); // Errore!
```

Per farlo dobbiamo riconvertire l'oggetto in un oggetto del DOM:

```
var test = $('#test')[0]; console.log(test.className); 'test'
```

La sintassi è:

```
var domObject = $(jqueryObject)[0];
```

Questo passaggio è ancora più evidente nel caso dei metodi di jQuery che operano sugli elementi:

```
$('#test').click(function() {          var $test = $(this); // oggetto jQuery        var
test = this;  // oggetto DOM           console.log($test.attr('class')); // 'test'
        console.log(test.className);  // 'test'    });
```

In questo caso `this` può operare sia come riferimento ad un oggetto jQuery che ad un oggetto del DOM.

Usare i metodi del DOM con jQuery

Perchè dovremmo usare i metodi del DOM con jQuery se esistono già quelli nativi della libreria? Semplice: i metodi del DOM vengono implementati nativamente nei browser con linguaggi come C++, quindi sono notevolmente più veloci di quelli di jQuery.

Prendiamo ad esempio il metodo `getElementById()`:

```
var test = document.getElementById('test'); var $test = $('#test');
```

Apparentemente il risultato è identico, ma se analizziamo il codice della libreria noteremo che jQuery esegue un procedimento di scomposizione dell'espressione CSS passata al wrapper `$()`. Infatti se il browser supporta le Selectors API, jQuery utilizza quelle, altrimenti il controllo passa all'engine Sizzle per l'elaborazione dell'espressione CSS tramite espressioni regolari.

A questo punto si potrebbe dire: noi vogliamo utilizzare i metodi di jQuery. Come facciamo se utilizziamo il DOM? Semplice: è sufficiente passare l'espressione DOM al wrapper `$()`:

```
var $test = $(test); // ossia $(document.getElementById('test')
```

Pensiamo ad esempio al contesto dei selettori. Questa è una soluzione che io uso molto spesso:

```
var $p = $('p', document.getElementById('test'));
```

La migliore performance del DOM si rivela con il tipico esempio della proprietà `innerHTML` (non facente parte delle specifiche ufficiali ma comunque assegnata da tutti i browser a ciascun oggetto di tipo `HTMLElement` che la accetti).

La controparte jQuery è `html()`, che però è meno performante della prima. Esempio:

```
var html = ''; var test = document.getElementById('test');  for(var i = 0; i < 1000; i +=
1) {    html += '<p>' + i + '</p>';   }  test.innerHTML = html;
```

`innerHTML` è più veloce perchè i browser utilizzano il loro parser HTML quando usiamo questa proprietà. E dato che in assoluto un parser HTML è forse il componente più veloce di un browser, è chiaro che questa proprietà è da preferire ovunque sia possibile utilizzarla.

Riassumendo: i metodi del DOM sono da preferire se la vostra esigenza primaria è la performance delle vostre pagine. Se avete ad esempio una tabella con 5000 righe, forse sarebbe opportuno valutare le possibilità offerte dal DOM.

Attenzione comunque: alcuni metodi del DOM, come ad esempio quelli relativi alla creazione di elementi e al loro popolamento, possono non essere così efficaci come i metodi di jQuery, soprattutto se pensiamo in termini di verbosità del codice. Esempio:

```
var test = document.getElementById('test'); var txt = document.createTextNode('Test');
test.appendChild(txt);
```

In jQuery invece:

```
$('#test').text('Test');
```

In questo caso risparmiamo molto codice utilizzando i metodi di jQuery.

jQuery: a cosa serve il metodo live()

Il metodo `.live()` di jQuery serve a gestire principalmente gli eventi per quegli elementi che non sono ancora presenti nel DOM ma che verranno inseriti successivamente. Questo metodo, ovviamente, funziona anche per gli elementi già presenti, ma la sua vera utilità diventa chiara quando intere strutture vengono aggiunte alla pagina in modo dinamico.

La sintassi del metodo è la seguente:

```
$(elemento).live(evento, callback);
```

`evento` è il nome di un normale evento jQuery (`click`, `focus` ecc.), mentre `callback` è la funzione da associare all'evento specificato.

Esempio:

```
$('#trigger').live('click', function(e) {     e.preventDefault();            // ... continua
});
```

I casi d'uso tipici sono:

- Strutture DOM generate via AJAX (per esempio come stringhe HTML passate al wrapper `$()`).
- Struttura DOM generate da eventi successivi al caricamento del documento.

`.live()` funziona secondo il principio alla base dell'Observer Pattern. In pratica l'intero oggetto `document` viene monitorato per intercettare l'inserimento di nuovi elementi e per associare l'evento specificato al nuovo elemento inserito.

Inserire jQuery nel nostro sito

jQuery può essere inclusa in vari modi nelle nostre pagine. In questo articolo vedremo i diversi modi di includere jQuery e il corretto ordine da seguire affinchè la libreria funzioni al meglio.

Posizione nella pagina

jQuery può essere inclusa sia nell'elemento `head` che prima della chiusura dell'elemento `body`. La seconda soluzione è da preferirsi in quanto aumenta di molto la performance delle nostre pagine.

Infatti i browser non aspettano il caricamento di jQuery prima di visualizzare la pagina poichè in questo caso jQuery è già inclusa nel corpo della pagina. Esempio:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script> </body> </html>
```

Caricamento locale e remoto

jQuery può essere sia scaricata dal sito ufficiale e quindi inclusa localmente nelle nostre pagine sia caricata da remoto (come nell'esempio precedente).

Il caricamento remoto ha tra i suoi vantaggi il fatto che la copia di jQuery è già compressa e ottimizzata. Inoltre il download remoto avviene tramite un sistema CDN che permette di ridurre al minimo la latenza.

Tra gli svantaggi c'è il fatto che se ci sono problemi con il download remoto il nostro sito potrebbe essere penalizzato in termini di caricamento. Per questo gli sviluppatori adottano spesso la copia locale come valore di ripiego nel caso in cui quella remota non sia disponibile:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script> <script
type="text/javascript"> if(!window.jQuery && document.write('<script type="text/javascript"
src="assets/js/jquery.min.js"></script>')) </script> </body> </html>
```

Se si usa Google per caricare jQuery, l'URL di base è sempre `http://ajax.googleapis.com/ajax/libs/jquery/`. Quest'URL può essere seguito dal numero di versione di jQuery. Il numero `1` indica sempre l'ultima sottoversione disponibile per quel numero. Se invece avessimo specificato `1.6`, avremmo caricato invece l'ultima sottoversione disponibile della versione `1.6`.

Ovviamente è possibile caricare direttamente una sottoversione specificandola ad esempio come `1.7.1`.

Caricamento in WordPress

jQuery è già disponibile in WordPress. Dobbiamo solo caricarla nel nostro tema utilizzando la funzione `wp_enqueue_script()` nel file `functions.php` o nei file del tema (`header.php` o `footer.php`):

```
if(!is_admin()) {      wp_enqueue_script('jquery'); }
```

Qualora volessimo usare una copia di jQuery diversa da quella di WordPress dobbiamo procedere in questo modo nel file `functions.php`:

```
if(!is_admin()) {      wp_deregister_script('jquery');       wp_register_script('jquery',
'http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js', '1.0', true);
        wp_enqueue_script('jquery'); }
```

In questo caso abbiamo sostituito nel nostro tema la copia locale con quella remota di Google.

Caricamento del codice jQuery

Il codice jQuery può essere eseguito all'interno di due eventi specifici:
l'evento `load` dell'oggetto `window` e l'evento `ready` dell'oggetto `document`. Il primo esegue il codice quando il documento non è ancora completo, mentre il secondo è invocato nel momento in cui tutto il documento è stato caricato. Vediamo insieme i dettagli.

L'evento load

Possiamo utilizzare questo evento quando non abbiamo bisogno di interagire con la struttura del documento, per esempio al fine di effettuare un preload delle immagini:

```
$(window).load(function() {    // codice });
```

L'evento ready

Questo evento va usato quando dobbiamo interagire con la struttura del documento e quindi abbiamo bisogno che questo sia completo:

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

Questo evento può anche essere scritto in forma abbreviata:

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

Combinare i due eventi

La priorità degli eventi è:

1. `load`
2. `ready`

Per usarli insieme dobbiamo utilizzare il seguente codice:

```
$(window).load(function() {    // codice su load()           $(function() {
        // codice su ready()           });  });
```

Gli eventi load e ready in WordPress

In WordPress jQuery è già nel flusso dell'applicazione, quindi occorre inserire i nostri eventi all'interno della seguente funzione self-executing che crea una sandbox attorno al nostro codice:

```
(function($) {         // qui vanno load() e ready()  })(jQuery);
```

Concetti fondamentali JavaScript per usare jQuery

In questo articolo vedremo alcuni concetti chiave della programmazione JavaScript che si rivelano essere fondamentali quando si utilizza jQuery. Ovviamente la trattazione non può essere esaustiva ma deve essere considerata come un'introduzione all'argomento.

Variabili

Le variabili, definite con la parola chiave `var`, in JavaScript contengono dati. JavaScript non obbliga il programmatore a dichiarare il tipo di dati contenuto in una variabile, perchè è l'interprete JavaScript a farlo durante l'esecuzione del codice.

Ecco alcuni tipi di variabili:

```
var str = 'Test'; // stringa var n = 1; // numero var is = true; // booleano var arr = [1,
2, 3]; // array var obj = {};  // oggetto (letterale) var $test = $('#test'); //
oggetto/elemento jQuery var test = document.getElementById('test'); // oggetto/elemento DOM
var re = /^\d+$/; // espressione regolare
```

Essendo un linguaggio basato sugli oggetti, è possibile invocare su questi tipi di dati il metodo corrispondente dell'oggetto base di riferimento. Ad esempio la variabile `str` fa riferimento all'oggetto base `String` perchè è una stringa.

Quindi se volessimo convertire questa stringa in caratteri maiuscoli dovremmo scrivere:

```
var strUpper = str.toUpperCase(); // 'TEST'
```

Oppure direttamente:

```
var str = 'Test'.toUpperCase(); // 'TEST'
```

Le variabili che abbiamo visto sinora sono dette **variabili globali** perchè sono visibili in tutto il nostro codice. Esistono tuttavia delle **variabili locali** che sono visibili solo in una determinata sezione in cui vengono definite. L'esempio classico è quello delle variabili create all'interno di funzioni:

```
var URL = location.href;  var getBaseURL = function() {      // qui URL è visibile
        var base = location.protocol + location.host;                // base non è
accessibile al di fuori della funzione getBaseURL();         return URL.replace(base, '');
};  alert(base); // Errore: undefined
```

`base` è accessibile solo all'interno del suo contesto, e se proviamo ad accedervi dall'esterno otteniamo il valore `undefined` che l'interprete JavaScript riserva a quelle variabili non definite.

Ecco un altro esempio:

```
var testFunc = function() {    var str = 'lorem';            var upperFirst = function() {
            // Qui str è visibile                      var first = str.charAt(0); // 'l',
visibile solo all'interno di upperFirst()             var upper = first.toUpperCase(); //
'L'             var output = upper + str.substring(1, str.length); // 'Lorem'
```

```
            return output; // 'Lorem'                    };      alert(first); //
Errore: undefined  };
```

Funzioni

Una funzione non è altro che un insieme di operazioni sui dati volto ad ottenere un risultato finale. Si usano le funzioni quando si vogliono manipolare i dati in modo pratico e riusabile.

Una funzione in JavaScript può essere definita in vari modi, tutti accomunati dalla parola chiave `function`:

```
function test() {      // codice  }  var test = function() {         // codice  };
(function() {  // codice  })();  test.click(function() {     // codice  });
```

Una funzione viene generalmente invocata in questo modo:

```
test();
```

Le funzioni accettano argomenti, che vengono passati in questo modo:

```
var multiply = function(a, b) { // a e b sono gli argomenti, separati da virgola    return
(a * b);  };  alert(multiply(3, 2)); // 6
```

Se una funzione restituisce un valore, come nel caso della moltiplicazione, allora si usa l'istruzione `return`. Questa istruzione può anche esseree utilizzata per interrompere l'esecuzione di una funzione:

```
var multiply = function(a, b) {       if(a == 0 || b == 0) {                return false;
}       return (a * b);  };
```

Selettori jQuery: panoramica

I selettori jQuery sono interamente basati sulla sintassi CSS. Questo permette anche ai non sviluppatori di intuirne immediatamente il significato. Alcuni di questi selettori non appartengono alle specifiche CSS ma sono specifici di jQuery. Vediamone insieme una panoramica.

Selettori

Selettore	Esempio	Seleziona
`*`	$("*")	Tutti gli elementi
`#id`	$("#lastname")	L'elemento con ID uguale a `lastname`
`.class`	$(".intro")	Tutti gli elementi con classe `intro`
`element`	$("p")	Tutti gli elementi `p`
`.class.class`	$(".intro.demo")	Tutti gli elementi con le classi `intro` e `demo`
`:first`	$("p:first")	Il primo elemento `p`
`:last`	$("p:last")	L'ultimo elemento `p`
`:even`	$("tr:even")	Tutti gli elementi pari
`:odd`	$("tr:odd")	Tutti gli elementi dispari
`:eq(`*`indice`*`)`	$("ul li:eq(3)")	Il quarto elemento in una lista (l'indice parte da 0)

`:gt(n)`	$("ul li:gt(3)")	Elementi della lista con un indice maggiore di 3
`:lt(n)`	$("ul li:lt(3)")	Elementi della lista con un indice minore di 3
`:not(selettore)`	$("input:not(:empty)")	Tutti gli input non vuoti
`:header`	$(":header")	`h1, h2, h3, h4, h5, h6`
`:animated`	$(":animated")	Tutti gli elementi animati
`:contains(testo)`	$(":contains('Test')")	Tutti gli elementi che contengono il testo specificato
`:empty`	$(":empty")	Tutti gli elementi senza elementi discendenti
:hidden	$("p:hidden")	Tutti gli elementi `p` visibili
`:visible`	$("table:visible")	Tutte le tabelle visibili
`s1,s2,s3`	$("th,td,.intro")	Tutti gli elementi che corrispondono ai selettori
`[attributo]`	$("[href]")	Tutti gli elementi con attributo `href`
`[attributo=valore]`	$("[href='default.htm']")	Tutti gli elementi con attributo `href` uguale a `default.htm`
`[attributo!=valore]`	$("[href!='default.htm']")	Tutti gli elementi con attributo `href` diverso da `default.htm`
`[attributo$=valore]`	$("[href$='.jpg']")	Tutti gli elementi il cui valore di attributo `href` termina con `.jpg`
`[attribute^=value]`	$("[href^='jquery_']")	Tutti gli elementi il cui valore di attributo `href` inizia con `jquery_`
`:input`	$(":input")	Tutti gli elementi di input
`:text`	$(":text")	`<input type="text"/>`
`:password`	$(":password")	`<input type="password"/>`
`:radio`	$(":radio")	`<input type="radio"/>`
`:checkbox`	$(":checkbox")	`<input type="checkbox"/>`
`:submit`	$(":submit")	`<input type="submit"/>`
`:reset`	$(":reset")	`<input type="reset"/>`
`:button`	$(":button")	`<input type="button"/>`
`:image`	$(":image")	`<input type="image"/>`
`:file`	$(":file")	`<input type="file"/>`
`:enabled`	$(":enabled")	Tutti gli elementi di input attivi
`:disabled`	$(":disabled")	Tutti gli elementi di input disattivati
`:selected`	$(":selected")	Tutti gli elementi di input selezionati
`:checked`	$(":checked")	Tutti gli elementi di input con un segno di spunta

Documentazione

1. Documentazione ufficiale
2. jQAPI

I metodi di jQuery

Questo articolo fornirà una panoramica completa dei principali metodi della libreria jQuery con particolare riferimento alla documentazione ufficiale dellaa libreria stessa.

Metodi jQuery

1. Core
2. Selettori
3. Gestione degli attributi HTML
4. Attraversamento del DOM

5. Manipolazione del DOM
6. Gestione delle proprietà CSS degli elementi
7. Eventi
8. Effetti
9. AJAX
10. Utility

Per approfondire si consulti la Guida jQuery di *Html.it*.

La catena di selettori in jQuery

Per *catena di selettori* (selector chain) in jQuery si intende il modo con cui la selezione iniziale degli elementi viene estesa aggiungendovi ulteriori selezioni e metodi che verranno eseguiti in serie da jQuery. Questo articolo si occuperà di questo aspetto.

La catena di selettori ha questa struttura di base:

```
$(selettore/i).metodo1().metodo2().metodoN();
```

Si inizia sempre con il metodo/wrapper `$()` (o `jQuery()`) a cui viene passato un selettore o più selettori e si prosegue quindi con una serie di metodi.

Ad esempio, ad un elemento con ID `test` possiamo aggiungere una classe CSS e quindi cambiarne il colore del testo con un'unica espressione:

```
$('#test').addClass('test').             css('color', 'blue');
```

La catena di selettori, specie se molto lunga, non andrebbe **mai** scritta su una sola riga, ma sempre suddivisa su righe successive come nell'esempio precedente.

Inoltre, per migliorare la performance, è sempre bene mettere in cache il selettore iniziale utilizzando una variabile:

```
var $test = $('#test'); $test.addClass('test').        css('color', 'blue');
```

Il wrapper $() di jQuery

Il wrapper `$()` di jQuery non è altro che un alias per `jQuery()`. Questo wrapper è alla base dell'intero meccanismo di selezione degli elementi in jQuery. Vediamolo in dettaglio.

L'uso principale di questo wrapper è la trasformazione di espressioni CSS in oggetti jQuery:

```
var items = $('#nav li');
```

Ora la variabile `items` contiene ciò che viene definito come un *wrapped set* di elementi. Questi elementi sono oggetti jQuery. Trattandosi di oggetti jQuery non è possibile operare su di essi

utilizzando i metodi e le proprietà del DOM. Per farlo occorre riportare questi elementi al loro stato originario:

```
var nav = $('#nav')[0]; // elemento DOM nav.className = 'nav';
```

Per gli elementi singoli si usa la notazione `[0]` degli array per trasformare l'elemento jQuery in un elemento DOM. Per elementi multipli si utilizza invece il metodo `.get()`:

```
var items = $('#nav li').get(); // elementi DOM  for(var i = 0; i < items.length; i++) {
        var item = items[i];   item.className = 'item-' + (i+1); }
```

Il wrapper `$()` accetta anche espressioni DOM per selezionare gli elementi:

```
var nav = $(document.getElementById('nav'));
```

L'espressione DOM viene convertita in modo tale che l'oggetto restituito sia un oggetto jQuery. Questo wrapper accetta anche oggetti JavaScript come argomento, anche se l'uso principale di questa tecnica è limitato alla creazione di eventi custom sugli oggetti.

Il wrapper `$()` può accettare come secondo parametro il *contesto* in cui vanno selezionati gli elementi. Per contesto si intende la sezione del documento a cui fanno riferimento gli elementi selezionati. Ad esempio:

```
var items = $('li', '#nav'); // #nav come contesto
```

Usare il contesto è altamente consigliato per migliorare la performance di jQuery. Non specificandolo, infatti, si obbliga jQuery ad effettuare una ricerca sull'intero oggetto `document`.

Infine, questo wrapper può essere usato per creare nuovi elementi. In questo caso si può specificare un oggetto letterale come secondo parametro che conterrà le proprietà e gli attributi del nuovo elemento:

```
$('<div/>', {  id: 'test',    'class': 'test'        }).appendTo('body');
```

Si tenga presente che il nuovo elemento non viene automaticamente inserito nel documento. Occorre sempre usare uno dei metodi di jQuery preposti per lo scopo.

I metodi di attraversamento del DOM di jQuery

jQuery dispone di diversi metodi per l'attraversamento del DOM. Attraversare il DOM significa muoversi lungo l'albero del documento e selezionare elementi. Vediamo in dettaglio i metodi più importanti.

prev() e next()

Come suggerisce il loro nome, questi due metodi selezionano rispettivamente l'elemento immediatamente precedente e immediatamente successivo ad un dato elemento.

Per esempio, data la seguente struttura:

```
<div id="test">        <p id="one">...</p>    <p id="two">...</p>    <p id="three">...</p>
</div>
```

ecco come possiamo usare questi due metodi:

```
var element = $('#two', '#test'); var first = element.prev(); // #one var last =
element.next(); // #three
```

Questi metodi accettano come parametro un selettore con cui è possibile selezionare un elemento specifico:

```
var element = $('#three', '#test'); var first = element.prev('#one'); // #one
```

prevAll() e nextAll()

Questi due metodi hanno un funzionamento identico a quelli visti in precedenza, con la differenza che invece di restituire un solo elemento restituiscono un nuovo set di elementi.

Esempio:

```
var element = $('#three', '#test'); var firstSecond = element.prevAll(); // #one, #two
```

prevUntil() e nextUntil()

Anche questi due metodi restituiscono un nuovo set di elementi, ma la selezione arriva fino (e non oltre) il selettore specificato come parametro (da cui il termine *until*, fino a).

Per esempio, data la seguente struttura:

```
<div id="test">        <p id="one">...</p>    <p></p>        <p></p>        <p></p>
       <p id="two">...</p> </div>
```

possiamo operare questa selezione:

```
var element = $('#one', '#test'); var nextPs = element.nextUntil('#two');  // #one, p, p, p
var element2 = $('#two', '#test'); var prevPs = element2.prevUntil('#one');  // #two, p, p,
p
```

parent() e parents()

Questi due metodi selezionano rispettivamente l'immediato genitore di un elemento e l'antenato di tale elemento. Per antenato si intende un elemento genitore che non contiene direttamente l'elemento selezionato ma che è invece un capostipite della struttura nella quale tale elemento risiede.

Esempio di una tale struttura è il seguente:

```
<div id="ancestor">    <div id="parent">             <div id="child"></div>        </div>
</div>
```

Ecco come usare questi metodi:

```
var child = $('#child'); var parent = child.parent(); // #parent var ancestor =
child.parents('#ancestor'); // #ancestor
```

siblings()

Questo metodo restituisce tutti gli elementi fratelli di un dato elemento. Un elemento è fratello di un altro quando entrambi hanno lo stesso genitore. Il metodo accetta come parametro un selettore per restituire solo un determinato tipo di elementi e non tutti.

Esempio:

```
var element = $('#one', '#test'); var siblings = element.siblings(); // #one, p, p, p var
siblingsFiltered = element.siblings('p:not([id])); // p, p, p
```

children()

Questo metodo restituisce tutti gli elementi figli di un dato elemento. Il metodo accetta come parametro un selettore per restituire solo un determinato tipo di elementi e non tutti.

Esempio:

```
var element = $('#test'); var pS = element.children('p:not([id])'); // p, p, p
```

I metodi di manipolazione del DOM di jQuery

jQuery dispone di diversi metodi per la manipolazione del DOM. Per manipolazione del DOM si intende l'inserimento di nuovi elementi, la loro cancellazione e posizionamento nella struttura del documento. In questo articolo vedremo i metodi più significativi.

insertBefore() ed insertAfter()

Per inserire un elemento nella pagina si utilizzano in genere questi due metodi i quali inseriscono il nuovo elemento rispettivamente prima e dopo l'elemento passato come argomento.

Ipotizzando la seguente struttura:

```
<div id="test">        <div id="target"></div> </div>
```

Utilizzando `insertBefore()` in questo modo:

```
$('<p/>').insertBefore('#target');
```

avremo:

```
<div id="test">        <p></p>        <div id="target"></div> </div>
```

Se invece utilizziamo `insertAfter()` avremo:

```
<div id="test">        <div id="target"></div>        <p></p> </div>
```

append(), appendTo(), prepend(), prependTo()

Questi metodi aggiungono un elemento alla struttura dell'elemento di destinazione prima (prepend) o dopo (append) il contenuto dell'elemento. La sintassi cambia in base al metodo:

- `$(elemento).append|prepend(nuovoElemento)`
- `$(nuovoElemento).appendTo|prependTo(elemento)`

Esempio:

```
$('<p/>').appendTo('#target'); $('#target').prepend('<p/>');
```

che genererà la seguente struttura:

```
<div id="test">        <div id="target">             <p></p>                <--contenuto-->
               <p></p>        </div> </div>
```

wrap(), wrapAll(), wrapInner()

Il verbo wrap in inglese indica il racchiudere qualcosa in qualcos'altro. Letteralmente significa "avvolgere", ed è esattamente quello che questi metodi fanno: essi racchiudono un elemento all'interno di un nuovo elemento HTML, ma con alcune differenze:

- `wrap()` racchiude un solo elemento
- `wrapAll()` racchiude più elementi
- `wrapInner()` racchiude il testo dell'elemento

Esempio:

```
$('#target').wrap('<div id="wrapper"></div>');
```

che genererà la seguente struttura:

```
<div id="test">   <div id="wrapper">  <div id="target"></div>   </div> </div>
```

clone()

Questo metodo clona un elemento, ossia ne crea una copia. Accetta come parametro un valore booleano che se viene impostato su `true` clona anche gli eventi associati all'elemento.

Un esempio tipico è il caso in cui vogliamo spostare un elemento in un'altra posizione nel DOM:

```
var copy = $('#target').clone(); $('#target').remove(); // rimuove l'elemento originale
$(copy).insertBefore('#test');
```

La struttura generata è la seguente:

```
<div id="target"></div> <div id="test"></div>
```

remove()

Questo metodo rimuove un elemento dal documento. Una volta rimosso, l'elemento non è più disponibile.

Leggere e scrivere gli stili CSS in jQuery

jQuery dispone dell'efficace metodo `.css()` per leggere e scrivere gli stili CSS di un elemento. In questo articolo vedremo in dettaglio questo metodo.

Questo metodo ha due sintassi distinte, una per leggere il valore di una proprietà CSS e l'altra per impostare gli stili dell'elemento. La prima è come mostrato nell'esempio:

```
var fontFamily = $('#test').css('fontFamily');
```

Come regola di massima le proprietà CSS che contengono trattini andrebbero scritte nella notazione camel case. Quindi `font-family` diventa `fontFamily`.

La seconda sintassi, in scrittura, è duplice. Possiamo avere una coppia di valori quando dobbiamo impostare una sola proprietà:

```
$('#test').css('color', '#c00');
```

oppure un più efficace oggetto letterale quando le proprietà da impostare sono più d'una:

```
$('#test').css({       color: '#333', padding: '1em',        border: 'thick solid',
        backgroundColor: 'silver' });
```

Le virgolette intorno alle proprietà CSS sono obbligatorie solo nel caso della proprietà `float` che in JavaScript è una parola riservata per uso futuro:

```
$('#test').css({       'float': 'left',       margin: '5px' });
```

Concatenare le animazioni in jQuery

La concatenazione di animazioni in jQuery avviene su gruppi di elementi. Funziona così: animiamo il primo elemento, lasciamo trascorrere un intervallo di tempo e così via per i successivi. Ci sono due modi per ottenere questo risultato: possiamo utilizzare la funzione che viene eseguita alla fine di un'animazione per creare l'animazione successiva o possiamo utilizzare la coda di animazioni con i metodi `queue()` e `dequeue()`. Vediamo insieme i dettagli.

Abbiamo questo gruppo di elementi:

```
<div id="wrapper">     <div class="box"></div>     <div class="box"></div>     <div
class="box"></div> </div>
```

Qui gli elementi sono dello stesso tipo, ma volendo possiamo usare la stessa classe CSS per animare elementi differenti. Il primo metodo è il seguente:

```
$(function() {         $('div.box:first').animate({          opacity: 0.5    }, 1000,
function() {                  $(this).next().animate({                        opacity: 0.5
});
```

Come si può notare abbiamo usato la funzione del metodo `animate()` per animare l'elemento successivo a quello corrente. La soluzione funziona, ma il nostro codice tende a diventare troppo prolisso.

La seconda soluzione, invece, crea una coda di animazioni con il metodo `queue()`. Il problema con questo metodo è che le animazioni partono simultaneamente, quindi dobbiamo utilizzare il metodo `delay()` con un timer incrementale per ovviare al problema:

```
$(function() {         var timer = 0;     $('div.box').each(function() {         var $box =
$(this);         $box.queue('opacity', function(next) {
$box.delay(timer).animate({                  opacity: 0.5             }, 1000, next);
});         $box.dequeue('opacity');          timer += 1000;     }); });
```

Il timer viene incrementato di 1000 millisecondi ad ogni iterazione del metodo `each()`. Il metodo `queue()` accetta due parametri: il primo è il nome dato alla coda di animazioni e il secondo è la funzione da chiamare ogni volta. Il parametro `next` serve a mettere in coda l'iterazione sull'elemento successivo. Appena la coda corrente termina il suo ciclo, dobbiamo liberare memoria utilizzando il metodo `.dequeue()` che accetta come parametro il nome della coda che abbiamo definito.

Potete visionare l'esempio finale in questa pagina.

Il metodo $.getJSON() di jQuery

Il metodo `$.getJSON()` di jQuery serve a reperire un file JSON e ad effettuarne il parsing. Questo metodo è ampiamente usato nell'interazione con molte API presenti sul Web, a cominciare da quelle di Twitter. Vediamone insieme i dettagli.

La sintassi di questo metodo è la seguente:

```
$.getJSON(URL, function(oggettoJSON) {        // parsing });
```

L'URL può essere sia assoluto che relativo. Nel caso di URL remoti, è di fondamentale importanza che il file JSON remoto sia servito come JSONP (JSON with Padding), poichè i browser per impostazione predefinita sono vincolati alla regola secondo cui una richiesta AJAX debba avere luogo sullo stesso dominio.

L'oggetto JSON passato come parametro dalla funzione di callback e restituito da quest'ultima deve essere un valido oggetto JSON. Ad esempio:

```
{       "a": 1,         "b": "Test" }
```

Il parsing avviene in questo modo:

```
$.getJSON('test.json', function(json) {       var a = json.a;         var b = json.b;
        console.log(a); // 1   console.log(b); // 'Test'  });
```

Come si può notare, abbiamo a che fare con un semplice oggetto JavaScript. Nel caso di JSONP la sintassi dell'URL è diversa:

```
var apiURL =
'https://api.twitter.com/1/users/show.json?screen_name=gabromanato&callback=?';
$.getJSON(apiURL, function(data) {    var followers = data.followers_count;
        console.log(followers); // es. 584  });
```

In questo caso il parametro di callback (`callback=?`) è fondamentale affinchè la richiesta abbia successo.

Il metodo load() di jQuery

Il metodo `load()` di jQuery è un metodo AJAX che permette di caricare del contenuto HTML esterno in un elemento della pagina corrente. Vediamone insieme i dettagli.

Supponiamo di avere un file HTML esterno così strutturato:

```
<h2>Test</h2> <p id="test">...</p>
```

Possiamo caricare l'intera struttura in un elemento in questo modo:

```
$('#wrapper').load('test.html');
```

Ora il nostro elemento apparirà così:

```
<div id="wrapper">     <h2>Test</h2>  <p id="test">...</p> </div>
```

Con questo metodo possiamo anche specificare l'elemento HTML esterno da caricare aggiungendo un selettore dopo il nome del file HTML:

```
$('#wrapper').load('test.html #test');
```

Ora il nostro elemento apparirà così:

```
<div id="wrapper">     <p id="test">...</p> </div>
```

Se usate degli stili CSS nella pagina esterna, non sempre questi verranno caricati nella pagina di destinazione. Infatti non tutti i browser sono in grado di clonare gli stili presenti nel file esterno. Per questo motivo se ne sconsiglia l'uso.

I metodi slideDown(), slideUp() e slideToggle() di jQuery

jQuery possiede tre distinti metodi per gestire lo sliding degli elementi: `slideDown()`, `slideUp()` e `slideToggle()`. Ciascuno di questi tre metodi crea un effetto diverso sugli elementi a seconda del fatto che l'elemento selezionato sia visibile o meno. Per visibilità in jQuery si intendono principalmente gli effetti ottenibili tramite le proprietà CSS `display` e `visibility`. Vediamo in dettaglio questi tre metodi.

Sintassi comune

```
$(elemento).slideNome(durata, callback)
```

- `durata`: la durata dell'animazione, espressa sia con parole chiave (`fast`, `normal`, `slow`, predefinito `normal`) che con un valore intero in millisecondi
- `callback`: la funzione da eseguire quando l'animazione è completa.

slideDown()

Questo metodo mostra un elemento nascosto facendolo scivolare dall'alto verso il basso:

```
down.click(function() {     test.slideDown(1000, function() {         alert('Complete');
}); });
```

slideUp()

Questo metodo nasconde un elemento visibile facendolo scivolare dal basso verso l'alto:

```
up.click(function() {     test.slideUp(1000, function() {         alert('Complete');
}); });
```

slideToggle()

L'azione di questo metodo varia in base alla visibilità dell'elemento: se l'elemento è nascosto, l'azione sarà identica a quella di `slideDown()`, se invecee l'elemento è visibile, l'azione sarà quella di `slideUp()`:

```
toggle.click(function() {     test.slideToggle(1000, function() {
alert('Complete');     }); });
```

Potete visionare l'esempio finale in questa pagina.

Il metodo attr() di jQuery

Il metodo `attr()` di jQuery serve a leggere e ad impostare gli attributi di un elemento. Vediamolo in dettaglio.

Sintassi di base

Lettura

```
$(elemento).attr(nomeattributo)
```

Scrittura

```
$(elemento).attr(nomeattributo, valore)

$(elemento).attr({attributo1: valore1, attributo2: valore2...})
```

Come si può notare, la sintassi è diversa a seconda se si usa questo metodo in lettura e scrittura. In lettura il metodo restituisce il valore di un dato attributo:

```
var id = $('#test').attr('id');
```

In scrittura invece la sintassi cambia se si vuole impostare uno o più attributi. Con un solo attributo si può impostare l'attributo in questo modo:

```
$('#test').attr('title', 'Test');
```

Se invece si vogliono impostare più attributi si può usare un oggetto letterale:

```
$('#test').attr({      title: 'Test', 'class': 'test });
```

`class` è una parola riservata in JavaScript, quindi occorre inserire il nome dell'attributo tra virgolette o apici.

I metodi fadeIn(), fadeOut(), fadeToggle() e fadeTo() di jQuery

jQuery dispone di quattro metodi per gestire l'opacità degli elementi: `fadeIn()`, `fadeOut()`, `fadeToggle()` e `fadeTo()`. Vediamoli in dettaglio.

Sintassi generale

fadeIn(), fadeOut(), fadeToggle()

```
$(elemento).fadeNone(durata, callback)
```

- `durata`: la durata dell'animazione, espressa o in parole chiave (`normal`, `fast`, `slow`, predefinito `normal`) o in millisecondi
- `callback`: la funzione da eseguire quando l'animazione è completa.

fadeTo()

```
$(elemento).fadeTo(durata, opacità, callback)
```

- `durata`: la durata dell'animazione, espressa o in parole chiave (`normal`, `fast`, `slow`, predefinito `normal`) o in millisecondi
- `opacità`: il valore di destinazione su cui impostare l'opacità finale dell'elemento
- `callback`: la funzione da eseguire quando l'animazione è completa.

fadeIn()

Questo metodo crea un effetto di assolvenza:

```
$in.click(function() {     test.fadeIn(1000, function() {         alert('Complete');
}); });
```

fadeOut()

Questo metodo crea un effetto di dissolvenza:

```
out.click(function() {     test.fadeOut(1000, function() {         alert('Complete');
}); });
```

fadeToggle()

Se l'opacità dell'elemento è diversa da 0, questo metodo crea un effetto di dissolvenza, in caso contrario di assolvenza:

```
toggle.click(function() {     test.fadeToggle(1000, function() {         alert('Complete');
}); });
```

fadeTo()

Questo metodo imposta l'opacità di un elemento su un valore dato:

```
to.click(function() {     test.fadeTo(1000, 0.5, function() {         alert('Complete');
}); });
```
[example-link url="http://jsfiddle.net/gabrieleromanato/cX88R/"]

Il metodo filter() di jQuery

Il metodo `filter()` di jQuery serve a filtrare un set di elementi in base ai parametri passati al metodo. Vediamone insieme i dettagli.

Sintassi di base

```
$(elementi).filter(selettore)
```

```
$(elementi).filter(funzione)
```

Si può passare al metodo sia un selettore che una funzione. La funzione accetta come parametro sia l'indice numerico dell'elemento corrente sia un riferimento all'elemento corrente.

Se si utilizza la funzione, questa deve sempre restituire (tramite `return`) un espressione booleana da valutare per filtrare gli elementi. Certamente utilizzare la funzione da molte più possibilità.

Ad esempio se volessimo selezionare solo i paragrafi che contengono uno username di Twitter, potremmo scrivere il seguente codice:

```
$('#filter').click(function() {     $('p').filter(function() {         var text = $(this).text();         return /(^|\s)@(\w+)/g.test(text);     }).addClass('filter'); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/djhvb/"]

I metodi show() e hide() di jQuery

jQuery dispone di due metodi per animare la proprietà CSS `display` di un elemento, ossia `show()` e `hide()`. Vediamo in dettaglio questi due metodi.

Sintassi di base

```
$(elemento).show|hide(durata, callback)
```

- `durata`: la durata dell'animazione, espressa sia con parole chiave (`normal`, `fast`, `slow`, predefinito `normal`) che in millisecondi
- `callback`: la funzione che viene eseguita quando l'animazione è completa.

show()

Questo metodo porta il valore CSS `none` della proprietà `display` fino al valore predefinito per l'elemento selezionato:

```
show.click(function() {     test.show(1000, function() {         alert('Complete');     }); });
```

hide()

Questo metodo porta il valore CSS predefinito della proprietà `display` fino a `none` per l'elemento selezionato:

```
hide.click(function() {     test.hide(1000, function() {         alert('Complete');     }); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/Ekd2L/"]

Il metodo slice() di jQuery

Il metodo `slice()` di jQuery serve ad ottenere una porzione di un set di elementi specificando l'indice di partenza e il numero di elementi da includere. Vediamolo in dettaglio.

Sintassi di base

```
$(elementi).slice(inizio, numeroelementi)
```

- `inizio`: l'indice parte da 0, quindi con 0 si parte dal primo elemento, con 1 dal secondo e così via
- `numeroelementi`: il numero di elementi da includere dopo l'elemento iniziale.

Esempio:

```
$('#slice').click(function() {     var set = $('li', '#test').slice(0, 4);
set.each(function() {         var text = $(this).text();
$(this).text(text.toLowerCase());     }); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/j5cpD/"]

Il metodo $.getScript() di jQuery

Il metodo `$.getScript()` di jQuery è un metodo AJAX che serve ad includere un file JavaScript esterno nel vostro codice e ad eseguirlo. Vediamonee insieme i dettagli.

Sintassi di base

```
$.getScript(URL, callback)
```

- `URL`: l'URL dello script da includere. L'URL deve appartenere allo stesso dominio dal quale si effettua la richiesta, perchè in questo caso si applica la policy AJAX sull'origine della richiesta.
- `callback`: la funzione che vi permette di utilizzare il codice contenuto nello script esterno.

Esempi

Supponiamo di avere questo script esterno:

```
var test = function() {        alert('Script esterno');  };
```

Possiamo usare il metodo in questo modo:

```
$.getScript('test.js', function() {   test(); // 'Script esterno'  });
```

Un altro uso, molto più interessante, è la possibilità di includere plugin jQuery al volo:

```
$.getScript('/scripts/jquery.color.js', function() {   $('#go').click(function(){
$('.block').animate( { backgroundColor: 'pink' }, 1000)       .delay(500)       .animate( {
backgroundColor: 'blue' }, 1000);   }); });
```

Il metodo serialize() di jQuery

Il metodo `serialize()` di jQuery viene applicato ai form e serve a costruire una query string (in genere POST) a partire dalle coppie `name` e `value` di ciascun elemento del form. Vediamolo in dettaglio.

Sintassi di base

```
$(form).serialize()
```

Un tipico esempio di utilizzo è l'invio di un form tramite AJAX:

```
$('#contact').submit(function(e) {    e.preventDefault();           var query =
$(this).serialize();          $.ajax({               type: 'POST',          url:
'ajax.php',            dataType: 'text',             data: query,           success:
function(msg) {                                      //...                          }
        });  });
```

Il metodo delay() di jQuery

Il metodo `delay()` di jQuery crea un ritardo tra un'animazione e la successiva nella catena di animazioni. Vediamolo in dettaglio.

Sintassi di base

```
delay(durata)
```

`durata`: la durata del ritardo tra un'animazione e l'altra espressa in millisecondi.

```
animate.click(function() {     test.fadeTo(800, 0.5).delay(2000).fadeTo(800, 1); });?
```
[example-link url="http://jsfiddle.net/gabrieleromanato/zb7D9/"]

Il metodo val() di jQuery

Il metodo `val()` di jQuery serve a leggere e scrivere il valore di un elemento, ossia, nello specifico, viene utilizzato per gli elementi dei form. Vediamolo in dettaglio.

Sintassi di base

Lettura

```
$(elemento).val()
```

Scrittura

```
$(elemento).val(valore)
```

Gli elementi dei form interessati sono sia quelli di tipo `input` che `textarea`. In scrittura si può utilizzare questo metodo per impostare un valore predefinito:

```
$('input#data-di-nascita').val('gg/mm/aaaa');
```

In lettura questo metodo viene spesso utilizzato per la validazione dei dati:

```
$('#email').blur(function() {         if($(this).val() == '') {                    //
errore         }  });
```

I metodi add(), eq(), end() ed andSelf() di jQuery

jQuery dispone dei metodi `add()`,`eq()`,`end()` and `andSelf()` per modificare i risultati restituiti dalla selezione operata dai selettori CSS. Vediamoli in dettaglio.

add()

Questo metodo accetta un selettore come argomento e aggiunge gli elementi selezionati alla selezione precedente:

```
$('#add').click(function() {     $('h1').add('h2').addClass('add'); });
```

eq()

Questo metodo seleziona un elemento specifico di una selezione utilizzando un indice numerico come argomento. L'indice parte da 0, che indica il primo elemento della selezione:

```
$('#eq').click(function() {     $('p').eq(1).addClass('eq'); });
```

end()

Questo metodo non ha argomenti. La sua funzione è quella di far tornare la selezione allo stato originario, ossia di ripassare il controllo al primo set della selezione:

```
$('#end').click(function() {     $('h1').next().addClass('end').end().addClass('end'); });
```

In questo caso il metodo riporta la selezione su `$('h1')`.

andSelf()

Questo metodo non ha argomenti. Il suo funzionamento fa in modo che la selezione corrente comprenda anche il selettore iniziale:

```
$('#andself').click(function() {
$('p').eq(0).next().addClass('andself').andSelf().addClass('andself'); });
```

In questo caso il metodo riporta la selezione su `$('p').eq(0)`.

[example-link url="http://jsfiddle.net/gabrieleromanato/X4Gkr/"]

Il metodo animate() di jQuery

Il metodo `animate()` di jQuery è il metodo principale per creare animazioni personalizzate. Vediamolo in dettaglio.

Sintassi di base

```
$(elemento).animate(stili, durata, easing, callback)
```

- `stili`: un oggetto letterale contenente i nomi e i valori delle proprietà CSS da animare
- `durata`: la durata dell'animazione, espressa in millisecondi
- `easing`: l'algoritmo di easing che regola la progressione dell'animazione. jQuery supporta nativamente i valori `swing` (predefinito) e `linear`, ma è possibile aggiungere altri valori, ad esempio con il plugin *jQuery Easing* o con `jQuery UI`
- `callback`: la funzione che viene eseguita quando l'animazione è completa.

Gli stili CSS supportati al meglio al momento sono:

1. larghezze e altezze
2. posizionamento
3. margini e padding
4. visibilità, con l'aggiunta dei valori custom `hide`, `show` e `toggle`
5. opacità

Nei valori CSS, questo metodo supporta una sintassi incrementale, ad esempio `'+=valore'` per aumentare il valore corrente e `'-=valore'` per diminuirlo.

I numeri senza unità di misura vengono interpretati come pixel in quelle proprietà CSS che supportano le unità di misura.

Esempio:

```
animate.click(function() {     test.animate({         left: 20,         padding: '1em',
opacity: 0.5,         margin: '1em 0',         width: '5em',         height: '5em'     },
1000, function() {         alert('Complete');     }); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/Aqdqa/"]

Il metodo removeAttr() di jQuery

Il metodo `removeAttr()` di jQuery serve a rimuovere un attributo da un elemento. Vediamolo in dettaglio.

Sintassi di base

```
$(elemento).removeAttr(attributo)
```

Esempio:

```
$('#test').removeAttr('title');
```

Calcolare le dimensioni degli elementi in jQuery

jQuery dispone di sei metodi distinti per calcolare la dimensione degli elementi utilizzando le regole del box model CSS. Vediamoli in dettaglio.

width() e height()

Questi due metodi restituiscono la larghezza e l'altezza di un elemento considerando solo l'area di contenuto, ossia senza padding e bordi:

```
$('#width').click(function() {     $('#output').text($('#test').width()); });
$('#height').click(function() {     $('#output').text($('#test').height()); });
```

innerWidth() ed innerHeight()

Questi due metodi restituiscono la larghezza e l'altezza di un elemento considerando anche il padding ma non i bordi:

```
$('#innerwidth').click(function() {     $('#output').text($('#test').innerWidth()); });
$('#innerheight').click(function() {     $('#output').text($('#test').innerHeight()); });
```

outerWidth() ed outerHeight()

Questi due metodi restituiscono la larghezza e l'altezza di un elemento considerando anche il padding ed i bordi:

```
$('#outerwidth').click(function() {     $('#output').text($('#test').outerWidth()); });
$('#outerheight').click(function() {     $('#output').text($('#test').outerHeight()); });
```

[example-link url="http://jsfiddle.net/gabrieleromanato/yRMHQ/"]

Uso dell'operatore this in jQuery

L'operatore `this` in JavaScript si riferisce all'oggetto corrente. Poichè gli elementi HTML vengono considerati come oggetti (o più tecnicamente nodi oggetto), questo operatore viene usato anche per gli elementi della pagina. jQuery utilizza la stessa sintassi, ma con alcune differenze. Vediamo quali.

Partiamo da un esempio JavaScript che sicuramente avrete già avuto modo di osservare su molti siti:

```
var button = document.getElementById('btn');  button.onclick = function() {
this.className = 'pressed';  };
```

In questo caso `this` fa riferimento a `button`, e quindi la proprietà `className` può essere usata utilizzando direttamente `this`.

Ora riscriviamo l'esempio con jQuery:

```
var button = $('#btn');  button.click(function() {      $(this).addClass('pressed');  });
```

`this` viene racchiuso all'interno del wrapper `$()` e per questo motivo non fa più riferimento ad un nodo oggetto (come nel primo esempio), ma è divenuto un riferimento ad un oggetto jQuery.

In questo caso la proprietà `className` non è più accessibile perchè non fa parte delle proprietà di un oggetto jQuery. Ma se facciamo un passo indietro e non usiamo il wrapper di jQuery, la proprietà è di nuovo accessibile:

```
var button = $('#btn');  button.click(function() {    this.className = 'pressed';      });
```

Un metodo alternativo è quello di usare la sintassi `$(elemento)[0]` che trasforma un oggetto jQuery in un normale nodo oggetto:

```
var button = $('#btn');  button.click(function() {    $(this)[0].className = 'pressed';
});
```

Questo risultato si può anche ottenere utilizzando la sintassi `$(elemento).get(0)`, o se sono più elementi `$(elementi).get()`.

Il metodo is() di jQuery

Il metodo `is()` di jQuery esegue un test booleano sull'elemento selezionato utilizzando un selettore come argomento. Vediamolo in dettaglio.

Sintassi di base

`$(elemento).is(selettore)`

Esempio:

```
if($('#test').is(':visible')) {      //...  }
```

Calcolare la posizione degli elementi con jQuery

jQuery dispone dei metodi `offset()` e `position()` per calcolare la posizione degli elementi nella pagina. Vediamoli in dettaglio.

offset()

Questo metodo restituisce le proprietà `top` e `left` calcolate rispetto all'intero documento. I valori di riferimento sono numeri interi.

```
$('#offset').click(function() {     var test = $('#test');     var top = test.offset().top;
var left = test.offset().left;     $('#output').text('top: ' + top + ', left: ' + left);
});
```

position()

Questo metodo restituisce le proprietà `top` e `left` calcolate rispetto all'elemento genitore che, di norma, dovrebbe essere posizionato in modo relativo. I valori di riferimento sono numeri interi.

```
$('#position').click(function() {     var test = $('#test');     var top =
test.position().top;     var left = test.position().left;     $('#output').text('top: ' +
top + ', left: ' + left); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/4jfhZ/"]

Domande frequenti (FAQ) su jQuery

In questo articolo affronteremo in dettaglio le domande più frequenti poste da chi utilizza jQuery per la prima volta.

Come posso selezionare un elemento usando una classe on un ID?

jQuery utilizza la stessa sintassi dei CSS. Quindi per un ID useremo:

```
$('#myDivId')
```

e per una classe:

```
$('.myCssClass')
```

oppure:

```
$('div.myCssClass')
```

esattamente come avviene nei CSS.

Come posso selezionare un elemento se ho già un elemento DOM?

Si può passare l'elemento DOM al wrapper `$()` che lo convertirà in un elemento jQuery:

```
var myDomElement = document.getElementById('foo');  $(myDomElement).find('a');
```

Come faccio a sapere se un elemento ha una particolare classe CSS?

Si può usare il metodo booleano `hasClass()`:

```
var $test = $('#test');  if($test.hasClass('evident')) {     //...  }
```

Per classi multiple si può utilizzare il metodo `is()`:

```
if($test.is('.evident.alert')) {      //... }
```

Come faccio a sapere se un elemento esiste?

Si può testare se l'elemento ha la proprietà `length` impostata:

```
if($('#test').length) {        //... }
```

Come faccio a sapere se un elemento è visibile o meno?

Si possono usare i selettori proprietari `:visible` e `:hidden`:

```
if($('#test').is(':hidden')) {        //...  }
```

Come faccio ad abilitare o disabilitare gli elementi dei form?

Si può usare il metodo `attr()` per impostare l'attributo `disabled` su `true`:

```
$('#submit').attr('disabled', true);
```

Per abilitarli si può rimuovere l'attributo `disabled`:

```
$('#submit').removeAttr('disabled');
```

Come faccio ad abilitare o disabilitare le checkbox?

Si può utilizzare lo stesso approccio visto in precedenza ma stavolta operando sull'attributo `checked`:

```
$('#choice').attr('checked', true); $('#choice').attr('checked', 'checked');
```

Stesso discorso se si vuole deselezionare una checkbox:

```
$('#choice').attr('checked', false); $('#choice').removeAttr('checked');
```

Come posso ottenere il valore selezionato di un elemento select?

Quando si invia un form, si può utilizzare direttamente il metodo `val()`:

```
var value = $('#select').val();
```

Durante la selezione, invece, si può usare l'evento `change()` e il selettore `:selected` sugli elementi `option`:

```
$('#select').change(function() {      var value = $('option:selected', $(this)).val();  });
```

Se invece si vuole selezionare il testo di un elemento `option` e non il suo valore, si può scrivere:

```
$('option:selected', '#select').text();
```

Come posso utilizzare la risposta del server in una richiesta AJAX?

Si possono utilizzare i metodi preposti per ciascun componente AJAX di jQuery:

```
$.ajax({      url: 'myPage.php',      success: function(response) {
alert(response); // la richiesta ha avuto successo      },      error: function(xhr) {
alert('Error!  Status = ' + xhr.status); // errore      } });
```

Per intervenire quando una richiesta è stata completata si può usare il metodo `complete`:

```
$.ajax({      url: 'myPage.php',      success: function(response) {
alert(response); // la richiesta ha avuto successo      },      error: function(xhr) {
alert('Error!  Status = ' + xhr.status); // errore      },      complete: function() {
        alert('Complete!'); // richiesta completata            } });
```

Come posso trasformare un elemento jQuery in un elemento DOM?

I seguenti due metodi sono equivalenti:

```
var test = $('#test')[0]; var test = $('#test').get(0);
```

Usare jQuery con altre librerie

Se si utilizza jQuery con altre librerie, l'unico problema che potrebbe sorgere è il conflitto derivante dall'uso del wrapper `$()` in quanto spesso usato anche da altre librerie (come Prototype). Vediamo come ovviare a questo problema.

La prima soluzione è quella di utilizzare il metodo `noConflict()` creando un nuovo alias per l'oggetto `jQuery`:

```
var j$ = jQuery.noConflict();  j$(document).ready(function() {       //...  });
```

La seconda soluzione è quella di creare un nuovo namespace attorno a jQuery utilizzando una funzione self-executing:

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

In questo modo il wrapper `$()` è isolato e non crea conflitti.

I metodi before() ed after() di jQuery

I metodi `before()` ed `after()` di jQuery accettano come parametro una stringa HTML e la inseriscono rispettivamente prima e dopo l'elemento selezionato. Vediamoli in dettaglio.

Sintassi

```
$(elemento).before|after(stringa)
```

La stringa HTML deve essere sempre ben formata per evitare di creare problemi nel DOM. Ad esempio, data la seguente struttura:

```
<div id="test"></div>
```

possiamo scrivere:

```
$('#test').before('<p>...</p>');
```

che produce questa nuova struttura:

```
<p>...</p> <div id="test"></div>
```

Gli eventi in jQuery

Un evento è un'azione che ha luogo nella pagina con o senza l'interazione diretta dell'utente. jQuery ci permette di eseguire del codice ogni volta che questa azione ha luogo tramite i suoi gestori di eventi. In questo articolo forniremo uno sguardo di insieme degli eventi in jQuery.

Un evento può essere associato direttamente ad un elemento utilizzando il metodo che porta il nome dell'evento:

```
$('#test').click(function() {         alert('Click');  });
```

All'interno della funzione l'elemento corrente viene indicato con `$(this)`:

```
$('#test').click(function() {         var $test = $(this);   alert($test.text());  });
```

La funzione utilizzata accetta come argomento un riferimento all'oggetto `event`, le cui proprietà e metodi più importanti sono:

- **`event.data`**: un oggetto letterale facoltativo passato alla funzione quando l'evento viene attivato sull'elemento
- **`event.namespace`**: il namespace specificato quando si innesca l'evento
- **`event.pageX`**: l'offset orizzontale del mouse rispetto al documento
- **`event.pageY`**: l'offset verticale del mouse rispetto al documento
- **`event.target`**: l'elemento DOM che ha creato l'evento
- **`event.type`**: il nome dell'evento
- **`event.preventDefault()`**: annulla l'azione predefinita associata all'evento (ad esempio sui link impedisce che questi aprano la risorsa specificata)
- **`event.which`**: per gli eventi associati al mouse ed alla tastiera, questa proprietà indica il tasto o il bottone premuto.

Esempio:

```
$('a').click(function(evt) {   evt.preventDefault();  });
```

Purtroppo con l'associazione diretta degli eventi si può usare un solo gestore di eventi alla volta. Ecco perchè si consiglia di utilizzare metodi come `bind()`, `on()` o `delegate()` quando si vogliono registrare più eventi sullo stesso elemento:

```
$('a').bind('click mouseover', function(e) {          var evtType = e.type;
        switch(evtType) {                    case 'click':
        e.preventDefault();                  break;          case 'mouseover':
```

Gli eventi possono anche essere lanciati senza un'azione diretta dell'utente:

```
$('#test').click(); $('a').trigger('click');
```

Di seguito riporto la documentazione ufficiale degli eventi in jQuery.

Documentazione

Events

Il metodo $.inArray() di jQuery

Il metodo `$.inArray()` è la versione jQuery del più celebre metodo di PHP. Il funzionamento è analogo: questo metodo restituisce un valore numerico (-1 in caso di insuccesso e 0 in caso di successo) che indica se un valore è presente nell'array. Vediamolo in dettaglio.

Sintassi

`$.inArray( valore, array [, indice] )`

- `valore`: il valore da cercare
- `array`: l'array dove cercare il valore
- `indice`: facoltativamente si può specificare l'indice da cui far partire la ricerca (il valore predefinito è 0).

Esempio:

```
var arr = ['a', 'b', 'c'];  if($.inArray('a', arr) != -1) {  // il valore è presente }
```

Il metodo $.get() di jQuery

Il metodo `$.get()` di jQuery viene utilizzato per effettuare una richiesta AJAX di tipo GET. Vediamolo in dettaglio.

Sintassi di base

`$.get(url, [query], [callback])`

- `url`: l'URL di riferimento per la richiesta
- `query`: i dati passati sia in formato stringa che come oggetto letterale
- `callback`: la funzione che viene eseguita quando la richiesta è stata completata e che ha come argomento i dati restituiti dal server.

Esempio di base:

```
$.get(   'ajax.php',   'mode=ajax',   function(data) {       alert(data);      } );
```

Volendo si può utilizzare un oggetto letterale per passare i dati al server:

```
$.get('ajax.php', { name: 'John', age: 35 },    function(data){      alert(data); });
```

Documentazione

jQuery.get()

Il metodo $.isArray() di jQuery

Il metodo `$.isArray()` è la versione jQuery del più celebre metodo di PHP. Il funzionamento è analogo: questo metodo restituisce un valore booleano che indica se l'argomento passato è un array. Vediamolo in dettaglio.

Sintassi

```
$.isArray(argomento)
```

Esempio:

```
var arr = ['a', 'b', 'c'];  if($.isArray(arr)) {     // arr è un array }
```

Il metodo $.ajax() di jQuery

Il metodo `$.ajax()` è considerato come un interfaccia AJAX di base da cui sono stati derivati altri metodi AJAX più evoluti. Nonostante questa definizione, questo metodo possiede una vasta gamma di opzioni che ci permettono di gestire ogni aspetto di una richiesta AJAX. Vediamolo in dettaglio.

Sintassi di base

```
$.ajax(opzioni)
```

Per un elenco completo delle opzioni disponibili, si consulti la documentazione ufficiale.

Esempi

Partiamo da un esempio base con le seguenti opzioni:

- `url`: l'URL a cui indirizzare la richiesta
- `type`: il tipo di richiesta (POST o GET)
- `dataType`: il tipo di dati restituito (HTML, JSON, JSONP, XML, testo semplice)
- `data`: i dati passati al server (sia sotto forma di query che di oggetto letterale)

- `success`: la funzione da eseguire quando la richiesta ha avuto successo.

```
$.ajax({       url: 'ajax.php',       type: 'POST',  dataType: 'html',      data:
$('#form').serialize(),       success: function(html) {
       $('#form').before(html);              } });
```

L'opzione `success` viene eseguita quando lo status dell'oggetto XMLHttpRequest restituisce un valore che indica il successo della richiesta. Se vogliamo invece eseguire il nostro codice quando la richiesta è stata completata, dobbiamo utilizzare l'opzione`complete`:

```
$.ajax({       url: 'ajax.php',       type: 'POST',  dataType: 'html',      data:
$('#form').serialize(),       success: function(html) {
       $('#form').before(html);              },     complete: function() {
```

Ovviamente il server potrebbe restituire un errore, per esempio con il codice 404 che indica che la risorsa non è stata trovata. In questo caso possiamo utilizzare l'opzione `error` che ha tra i suoi argomenti lo stesso oggetto XMLHttpRequest:

```
$.ajax({       url: 'ajax.php',       type: 'POST',  dataType: 'html',      data:
$('#form').serialize(),       success: function(html) {
       $('#form').before(html);              },     complete: function() {
```

Il parsing dei dati restituiti cambia ovviamente in base al tipo di dati. Per esempio se specifichiamo `json` come tipo di dati avremo:

```
$.ajax({       url: 'ajax.php',       type: 'POST',  dataType: 'json',      data:
$('#form').serialize(),       success: function(json) {
       $('#form').before('<p>' + json.message + '</p>');           },     complete:
function() {                  $('#form').slideUp(600);             },     error:
function(xhr) {                       console.log(xhr.status); // es. 404 Not Found
       } });
```

Il caso del tipo di dati `xml` è particolare: infatti ciò che viene restituito è un intero oggetto di tipo `DOMDocument`, l'equivalente del classico oggetto `document` a cui siamo abituati in HTML.

In questo caso siamo costretti ad usare il metodo `find()` per accedere agli elementi XML:

```
$.ajax({       url: 'ajax.php',       type: 'POST',  dataType: 'xml',       data:
$('#form').serialize(),       success: function(doc) {                     var items =
$(doc).find('item'),                  i,                        len = items.length;
       for(i = 0; i < len; i++) {                                    //...
              console.log(xhr.status); // es. 404 Not Found                } });
```

Come si può notare, il metodo `$.ajax()` offre un controllo ed un livello di dettaglio maggiore rispetto ad altri metodi AJAX di jQuery.

Il metodo $.post() di jQuery

Il metodo `$.post()` di jQuery viene utilizzato per effettuare una richiesta AJAX di tipo POST. Vediamolo in dettaglio.

Sintassi di base

```
$.post(url, [query], [callback])
```

- `url`: l'URL di riferimento per la richiesta
- `query`: i dati passati sia in formato stringa che come oggetto letterale
- `callback`: la funzione che viene eseguita quando la richiesta è stata completata e che ha come argomento i dati restituiti dal server.

Esempio di base:

```
$.post(   'ajax.php',   $('#form').serialize(),   function(data) {          alert(data);
} );
```

Volendo si può utilizzare un oggetto letterale per passare i dati al server:

```
$.post('ajax.php', { name: 'John', age: 35 },    function(data){      alert(data); });
```

Documentazione

jQuery.post()

Il metodo $.isPlainObject() di jQuery

Il metodo `$.isPlainObject()` serve a verificare se l'oggetto passato come argomento è stato creato come oggetto letterale o tramite la notazione `new Object()` restituendo un valore booleano. Vediamolo in dettaglio.

Sintassi

```
$.isPlainObject(oggetto)
```

Esempio:

```
var plain = {a: 'test'}; var func = function() {};  console.log($.isPlainObject(plain)); //
true console.log($.isPlainObject(func)); // false
```

Introduzione a jQuery per designers: le basi

jQuery è una potente libreria JavaScript che negli ultimi anni ha raggiunto un'elevatissima popolarità. Il segreto del suo successo sta nel fatto che jQuery rende semplice ciò che semplice non è, secondo il motto "fai di più, scrivi meno codice". In questo articolo spiegheremo come muovere i primi passi con jQuery, fornendo di fatto un'introduzione a questa libreria.

Installare jQuery

jQuery viene fornita in due formati: un formato compresso e minimizzato, più leggero e quindi più veloce da scaricare nel browser, il cui file prende il nome di `jquery-versione.min.js` e un formato per lo studio del sorgente non compresso nè minimizzato, e quindi più pesante dal nome `jquery-versione.js`, dove per *versione* si intende il numero di versione di jQuery.

Possiamo sia scegliere di scaricare tali file dal sito di jQuery oppure di usare un sistema CDN che contenga il file che ci interessa. Al momento il sistema CDN più popolare è quello di Google. Per inserire l'ultima versione di jQuery, possiamo usare questo snippet:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
```

Quando inseriamo jQuery dobbiamo fare in modo che la libreria venga prima dei nostri script nel sorgente, in questo modo:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script> <script
type="text/javascript" src="mio-script.js"></script>
```

Se non seguiamo ques'ordine, il browser restituirà degli errori JavaScript, in quanto jQuery verrebbe caricato dopo.

I selettori e la scorciatoia $

jQuery usa i selettori CSS, sia ufficiali che proprietari, per selezionare gli elementi nell'albero del documento. Per esempio in una struttura HTML come questa:

```
<ul id="test">   <li>A</li>   <li>B</li>   <li>C</li> </ul>
```

possiamo usare la catena di selettori di jQuery per esempio per operare sul primo elemento della lista. In jQuery, una catena di selettori inizia sempre con la scorciatoia $:

```
var A = $('#test li:first-child').text(); alert(A); // visualizza la stringa 'A'
```

Il codice precedente è suddiviso in tre parti:

1. l'alias/scorciatoia $ (riferimento a `jQuery`)
2. la catena di selettori (`(#test li:first-child)`)
3. un metodo che opera sull'elemento selezionato, in questo caso `text()`, che restituisce il nodo di testo dell'elemento.

La catena di selettori e metodi è virtualmente infinita. Per esempio, se avessimo voluto selezionare prima la prima voce di lista e aggiungervi una classe CSS e quindi la seconda voce con una seconda classe, avremmo scritto:

```
$('#test li:first-child') .addClass('test1') .next() .addClass('test2');
```

Ossia:

1. seleziona la prima voce (`$('#test li:first-child')`)
2. aggiungi la classe CSS `test1` (`addClass('test1')`)
3. spostati all'elemento successivo (`next()`)
4. aggiungi la classe CSS `test2` (`addClass('test2')`)

Come si può vedere, una conoscenza di base di HTML e CSS è sufficiente per cominciare ad usare jQuery.

Il documento è pronto?

JavaScript ha bisogno che tutto il documento sia caricato nella sua completezza per poter cominciare ad operare. Capita spesso infatti che un'immagine o un video blocchino l'esecuzione dei nostri script perchè la struttura del DOM non è ancora completa. jQuery risolve brillantemente il problema usando l'evento `ready()` associato all'oggetto `document`:

```
$(document).ready(function() {      // esecuzione del codice della pagina  });
```

In questo caso il nostro codice verrà eseguito quando effettivamente il documento è caricato, evitandoci problemi con i nostri script.

Altre caratteristiche di jQuery

Con jQuery si può inoltre:

1. gestire completamente gli eventi JavaScript
2. gestire completamente AJAX
3. creare animazioni, effetti visivi e altro ancora
4. gestire la programmazione orientata agli oggetti
5. lavorare in modo avanzato con gli array

e molto altro ancora...

La proprietà di jQuery jQuery.fx.interval

La proprietà interna di jQuery `jQuery.fx.interval` imposta il valore dei frame al secondo utilizzati nelle animazioni. Il valore predefinito è 13 millesimi di secondo. Non tutti i browser supportano correttamente questa proprietà che, nello specifico, può influenzare la performance del browser. Vediamo un esempio pratico.

```
jQuery.fx.interval = 4;  $('#run').click(function() {     $('#test').toggle(1000); });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/2Rejb/"]

Il metodo $.isEmptyObject() di jQuery

Il metodo `$.isEmptyObject()` serve a verificare se l'oggetto passato come argomento è privo di proprietà restituendo un valore booleano. Vediamolo in dettaglio.

Sintassi

```
$.isEmptyObject(oggetto)
```

Esempio:

```
var empty = {}; var obj = {a: 'test'};  console.log($.isEmptyObject(empty)); // true
console.log($.isEmptyObject(obj)); // false
```

CSS e animazioni jQuery: aspetti da considerare

Le animazioni jQuery utilizzano le proprietà CSS per poter operare. Quindi è di fondamentale importanza conoscere l'effetto che alcune proprietà CSS hanno sugli elementi al fine di evitare problemi nella visualizzazione.

Slideshow e proprietà overflow

Negli slideshow la proprietà CSS `overflow` serve a limitare la quantità di contenuto visibile. Per esempio:

```
#slideshow {    width: 500px;  overflow: hidden; }  #slides-wrapper {        width: 5000px;
        height: 400px; }
```

In questo caso la dichiarazione `overflow: hidden` limita la larghezza visibile a 500 pixel, anche se il contenitore delle slide è largo 5000 pixel. Questa dichiarazione ha però un effetto collaterale: se volessimo infatti posizionare due bottoni per la navigazione delle slide ai lati del contenitore, non potremmo farlo perchè essi verrebbero tagliati fuori in quanto posizionati all'esterno dei 500 pixel disponibili.

La soluzione in questo caso è quella di aggiungere un elemento intermedio tra il contenitore principale e quello delle slide:

```
<div id="slideshow">   <div id="slideshow-inner">            <div id="slides-wrapper"><!--
slide--></div> </div>  <a href="#" id="previous"><</a>       <a href="#" id="next">></a>
</div>
```

Quindi avremo:

```
#slideshow {    width: 500px;  position: relative; }  #slideshow-inner {     width: 500px;
        overflow: hidden; }  #slides-wrapper {        width: 5000px; height: 400px; }
#previous, #next {     position: absolute;    width: 20px;   height: 20px;  top: 50%;
        margin-top: -10px; }  #previous {     left: -20px; }  #next {        right: -20px; }
```

In questo caso i bottoni saranno visibili in quanto il contenitore principale non ha più la proprietà `overflow` impostata su `hidden`.

Posizionamento

I tipi di posizionamento più usati nelle animazioni sono:

- `position: relative`
- `position: absolute`

Nel primo tipo le coordinate dell'elemento vengono calcolate in base alla sua posizione corrente. Quindi se un elemento ha un margine sinistro di 15 pixel, l'animazione partirà da quei 15 pixel a sinistra.

Nel secondo tipo, invece, le coordinate vengono calcolate partendo dall'intera pagina. L'unica eccezione si ha quando il genitore dell'elemento animato ha la dichiarazione `position: relative` o `position: absolute`. In questo caso si parla di*posizionamento contestuale* e le coordinate verranno calcolate a partire dall'elemento genitore e non dall'intera pagina.

Floating

La proprietà CSS `float` viene spesso usata per allineare gli elementi su più righe. Usando questa proprietà, lo spazio verticale dell'elemento che contiene il float collassa, a meno che questo genitore non abbia un'altezza dichiarata o non venga usato un metodo per contenere il floating, come ad esempio `overflow: hidden`.

Se il genitore è posizionato relativamente, il metodo jQuery `position()` restituirà correttamente le coordinate di ciascun elemento flottato all'interno di tale genitore.

Per evitare che i float vadano su una nuova riga in modo casuale, è necessario calcolare lo spazio disponibile e ripartirlo su ciascun float. Se ad esempio abbiamo un contenitore largo 300 pixel e vogliamo tre float su ciascuna riga con uno spazio orizzontale e verticale tra ogni elemento, possiamo scrivere:

```
div.float {    width: 95px;   margin: 0 5px 5px 0;   float: left; }
```

Ossia 95 di larghezza più i 5 pixel del margine destro fanno 100 pixel e 300 diviso 3 fa appunto 100.

La proprietà di jQuery jQuery.fx.off

La proprietà interna di jQuery `jQuery.fx.off` se viene impostata su `true` annulla tutte le animazioni in corso portando l'effetto direttamente al suo stato finale. Questa proprietà può essere utilizzata per disabilitare temporaneamente le animazioni per motivi di accessibilità o di performance. Vediamone un esempio pratico.

```
var toggleFx = function() {     $.fx.off = !$.fx.off; }; toggleFx();
$('button').click(toggleFx)  $('input').click(function() {     $('div').toggle('slow');
});?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/uUVfC/"]

Il metodo $.isNumeric() di jQuery

Il metodo `$.isNumeric()` serve a verificare se l'argomento passato è di tipo numerico restituendo un valore booleano. Vediamolo in dettaglio.

Sintassi

```
$.isNumeric(valore)
```

Esempio:

```
$.isNumeric("-10");  // true $.isNumeric(16);     // true $.isNumeric(0xFF);   // true
$.isNumeric("0xFF"); // true $.isNumeric("8e5");  // true $.isNumeric(3.1415); // true
$.isNumeric(+10);    // true $.isNumeric(0144);   // true  $.isNumeric("");     // false
$.isNumeric({});     // false $.isNumeric(NaN);    // false $.isNumeric(null);   // false
$.isNumeric(true);   // false $.isNumeric(Infinity); // false $.isNumeric(undefined); //
false
```

Il metodo $.makeArray() di jQuery

Il metodo `$.makeArray()` converte qualsiasi oggetto simile ad un array in un vero array JavaScript. Vediamolo in dettaglio.

Sintassi

```
$.makeArray(oggetto)
```

Esempio:

```
var elems = document.getElementsByTagName('div'); // oggetto DOM NodeList var arr =
$.makeArray(elems); arr.reverse(); // usa un metodo degli array
```

Si tenga presente che se questo metodo viene usato sugli oggetti jQuery questi ultimi non possiederanno più i metodi jQuery a loro associati in quanto vengono di fatto trasformati in un array JavaScript:

```
var obj = $('li'); var arr = $.makeArray(obj);
```

Il metodo scrollLeft() di jQuery

Il metodo `scrollLeft()` restituisce o imposta il valore corrente della posizione orizzontale della scrollbar per un dato elemento. Vediamolo in dettaglio.

Sintassi

Lettura

```
$(elemento).scrollLeft()
```

Scrittura

```
$(elemento).scrollLeft(valore)
```

Tipicamente questo valore viene utilizzato come proprietà del metodo `animate()`:

```
$('#scroll').click(function() {     $('html,body').animate({         scrollLeft:
$('#test').css('left')     }, 800, function() {          $('html, body').animate({
scrollLeft: 0         }, 800);      }); });?
```

Per funzionare questo metodo ha bisogno che la proprietà CSS `overflow` **non** sia impostata su `hidden`.

[example-link url="http://jsfiddle.net/gabrieleromanato/afzVe/"]

Il metodo scrollTop() di jQuery

Il metodo `scrollTop()` restituisce o imposta il valore corrente della posizione verticale della scrollbar per un dato elemento. Vediamolo in dettaglio.

Sintassi

Lettura

```
$(elemento).scrollTop()
```

Scrittura

```
$(elemento).scrollTop(valore)
```

Tipicamente questo valore viene utilizzato come proprietà del metodo `animate()`:

```
$('#scroll').click(function() {     $('html,body').animate({         scrollTop:
$('#test').css('top')     }, 800, function() {          $('html, body').animate({
scrollTop: 0         }, 800);      }); });?
```

Per funzionare questo metodo ha bisogno che la proprietà CSS `overflow` **non** sia impostata su `hidden`.

[example-link url="http://jsfiddle.net/gabrieleromanato/ChDdE/"]

Il metodo offsetParent() di jQuery

Il metodo `offsetParent()` restituisce l'antenato più prossimo di un elemento che abbia la proprietà CSS `position` impostata su un valore diverso da `static` (predefinito). Vediamolo in dettaglio.

Avendo la seguente struttura HTML:

```
<div id="wrapper">     <div id="inner">         <div id="test"></div>     </div> </div>
```

Con i seguenti stili CSS:

```
#wrapper {     margin: 1em auto;     width: 600px;     position: relative;     border: 1px solid; }  #inner {     padding: 1em; }  #test {     width: 100px;     height: 100px; background: green; }
```

Avremo il seguente risultato:

```
$('#offset').click(function() {     $('#test').offsetParent().addClass('offset'); });?
```
[example-link url="http://jsfiddle.net/gabrieleromanato/58fQw/"]

Il metodo $.holdReady() di jQuery

Il metodo `$.holdReady()` permette di ritardare l'esecuzione dell'evento `ready` di jQuery. Vediamolo in dettaglio.

Questa caratteristica avanzata viene usata di solito quando si vogliono caricare script aggiuntivi (come i plugin) prima di lasciare che l'evento `ready` abbia luogo, anche se il DOM è completo.

Questo metodo deve essere invocato nell'elemento `head` subito dopo la libreria jQuery. Invocare il metodo dopo l'evento `ready` non sortisce alcun effetto.

Per ritardare l'evento `ready`, va prima usato `$.holdReady(true)`. Quando l'evento `ready` deve essere eseguito, si deve invocare `$.holdReady(false)`.

Si noti che questa procedura può essere ripetuta più volte per ogni chiamata a `$.holdReady(true)`. L'evento `ready` non avrà luogo fino a quando il parametro passato al metodo `$.holdReady()` non sarà `false`.

Esempio:

```
$.holdReady(true); $.getScript('my-plugin.js', function() {     $.holdReady(false); });
```

Il metodo html() di jQuery

Il metodo `html()` può essere usato per leggere e scrivere il contenuto HTML di un elemento. Vediamolo in dettaglio.

Sintassi di base

Lettura

```
$(elemento).html()
```

Scrittura

```
$(elemento).html(stringa)
```

```
$(elemento).html(function(indice, html))
```

In scrittura questo metodo accetta una funzione di callback con due parametri: `indice` rappresenta l'indice numerico dell'elemento all'interno del set di jQuery, mentre `html` il suo contenuto HTML da modificare.

La stringa HTML passata in scrittura dovrebbe sempre contenere marcatura ben formata:

```
$('div.demo-container')   .html('<p>All new content. <em>You bet!</em></p>');
```

Il codice di cui sopra crea la seguente struttura:

```
<div class="demo-container">   <p>All new content. <em>You bet!</em></p> </div>
```

Ecco invece un esempio con la funzione di callback:

```
$('div.demo-container').html(function() {   var emph = '<em>' + $('p').length + '
paragraphs!</em>';   return '<p>All new content for ' + emph + '</p>'; });
```

che può restituire:

```
<div class="demo-container">  <p>All new content for <em>6 paragraphs!</em></p>. </div>
```

Nota su Internet Explorer 8 e 9

Queste versioni di Explorer hanno dei problemi quando si sovrascrive il contenuto HTML di un elemento usando il metodo `html()`. La documentazione ufficiale raccomanda questo approccio:

```
$(elemento).empty().html(stringa);
```

Il metodo text() di jQuery

Il metodo `text()` può essere usato per leggere e scrivere il testo di un elemento. Per testo si intende l'insieme dei nodi DOM di tipo `text` che compongono l'elemento (inclusi i suoi discendenti). Vediamolo in dettaglio.

Sintassi di base

Lettura

```
$(elemento).text()
```

Scrittura

```
$(elemento).text(stringa)
```

```
$(elemento).text(function(indice, testo))
```

Ecco un esempio in lettura:

```
<div class="demo-container">   <div class="demo-box">Demonstration Box</div>   <ul>
<li>list item 1</li>   <li>list <strong>item</strong> 2</li>   </ul> </div>
```

Il codice `$('div.demo-container').text()` restituisce:

```
Demonstration Box list item 1 list item 2
```

In scrittura la funzione di callback accetta due parametri: l'indice dell'elemento nel set jQuery e il testo di tale elemento da modificare. Esempio:

```
$('ul li').text(function(index) {   return 'item number ' + (index + 1); });
```

Viene generata la seguente struttura:

```
<ul>   <li>item number 1</li>   <li>item number 2</li>   <li>item number 3</li> </ul>
```

Il metodo replaceWith() di jQuery

Il metodo `replaceWith()` sostituisce un elemento con un altro elemento. Questo elemento può essere sia già presente nel DOM che creato ex novo come stringa HTML. Vediamolo in dettaglio.

Sintassi di base

`$(elemento).replaceWith(elemento)`

`$(elemento).replaceWith(funzione)`

Partendo dalla seguente struttura:

```
<div class="container">   <div class="inner first">Hello</div>   <div class="inner
second">And</div>   <div class="inner third">Goodbye</div> </div>
```

Possiamo scrivere:

```
$('div.second').replaceWith('<h2>New heading</h2>');
```

E otterremo:

```
<div class="container">   <div class="inner first">Hello</div>   <h2>New heading</h2>
<div class="inner third">Goodbye</div> </div>
```

La sostituzione può essere applicata ad un set di elementi:

```
$('div.inner').replaceWith('<h2>New heading</h2>');
```

E otterremo:

```
<div class="container">   <h2>New heading</h2>   <h2>New heading</h2>   <h2>New
heading</h2> </div>
```

Possiamo anche utilizzare un selettore per la sostituzione:

```
$('div.third').replaceWith($('.first'));
```

E otterremo:

```
<div class="container">   <div class="inner second">And</div>   <div class="inner
first">Hello</div> </div>
```

Infine, possiamo utilizzare la funzione di callback:

```
$('div.container').replaceWith(function() {   return $(this).contents(); });
```

E otterremo:

```
<div class="inner first">Hello</div> <div class="inner second">And</div> <div class="inner
third">Goodbye</div>
```

Il metodo $.hasData() di jQuery

Il metodo `$.hasData()` restituisce un valore booleano che verifica se l'elemento passato come argomento possiede dei dati associati tramite il metodo `$.data()`. Vediamolo in dettaglio.

Sintassi di base

`$.hasData(elemento)`

Esempio:

```
$('#test').click(function() {     var $data = $('#data');     var $noData = $('#no-data');
$.data($data, 'test', 1);          alert($.hasData($data));  // true
alert($.hasData($noData));  // false });?
```

[example-link url="http://jsfiddle.net/gabrieleromanato/jB6rq/"]

Il metodo find() di jQuery

Il metodo `find()` serve ad effettuare ricerche all'interno della struttura DOM di un elemento. Restituisce l'elemento o gli elementi trovati come un nuovo set jQuery. Vediamolo in dettaglio.

Sintassi di base

`$(elemento).find(selettore|oggetto)`

Ad esempio, data la seguente struttura:

```
<ul class="level-1">   <li class="item-i">I</li>   <li class="item-ii">II     <ul
class="level-2">       <li class="item-a">A</li>       <li class="item-b">B         <ul
class="level-3">           <li class="item-1">1</li>           <li class="item-2">2</li>
<li class="item-3">3</li>         </ul>       </li>       <li class="item-c">C</li>
</ul>   </li>   <li class="item-iii">III</li> </ul>
```

Possiamo selezionare quelle voci di lista la cui classe non termini con la stringa `i`:

```
$('#find').click(function() {     $('ul.level-
1').find('li:not([class$="i"])').addClass('test'); });?
```
[example-link url="http://jsfiddle.net/gabrieleromanato/zAzz8/"]

Il metodo has() di jQuery

Il metodo `has()` serve a verificare, in modo booleano, se un elemento contiene un determinato elemento tra i suoi discendenti specificato come selettore. Vediamolo in dettaglio.

Sintassi di base

```
$(elemento).has(selettore)
```

Data ad esempio la seguente struttura:

```
<ul>   <li>list item 1</li>   <li>list item 2     <ul>       <li>list item 2-a</li>
<li>list item 2-b</li>     </ul>   </li>   <li>list item 3</li>   <li>list item 4</li>
</ul>
```

Possiamo selezionare solo le voci che contengono un'altra lista:

```
$('#has').click(function() {     $('li').has('ul').addClass('test'); });?
```
[example-link url="http://jsfiddle.net/gabrieleromanato/tVznn/"]

Il metodo $.param() di jQuery

Il metodo `$.param()` crea una versione serializzata di un oggetto JavaScript e ne codifica le parti in modo da renderlo adatto ad una richiesta AJAX. Vediamolo in dettaglio.

Sintassi di base

```
$.param(oggetto)
```

Esempio:

```
var myObject = {     a: {         one: 1,         two: 2,         three: 3     },     b:
[1, 2, 3] }; var recursiveEncoded = $.param(myObject); var recursiveDecoded =
decodeURIComponent($.param(myObject));  $('#encoded').click(function() {
alert(recursiveEncoded); });  $('#decoded').click(function() {     alert(recursiveDecoded);
});?
```

Che produce:

```
a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&b%5B%5D=2&b%5B%5D=3
```

```
a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3
```

[example-link url="http://jsfiddle.net/gabrieleromanato/D6R6N/"]

Il metodo toArray() di jQuery

Il metodo `toArray()` trasforma un set di elementi jQuery in un array JavaScript. Vediamolo in dettaglio.

Sintassi di base

```
$(elementi).toArray()
```

Esempio:

```
alert($('li').toArray());
```

Che produce:

```
[<li id="foo">, <li id="bar">]
```

I metodi get() e index() di jQuery

I metodi `get()` e `index()` servono rispettivamente per trasformare degli elementi jQuery in elementi DOM e per conoscere l'indice numerico di un elemento all'interno di un set. Vediamoli in dettaglio.

Sintassi di base

```
$(elemento|i).get()
```

```
$(elemento).index()
```

Esempi:

```
var domElements = $('li').get(); // DOM NodeList var $index = $('li:first + li').index();
// 1
```

Il metodo size() di jQuery è deprecato

Il metodo `size()`, in origine usato per reperire il numero di elementi presenti in un set jQuery, è stato deprecato. Vediamo i dettagli di questo problema.

La documentazione ufficiale afferma:

The `.size()` method is functionally equivalent to the `.length` property; however, the `.length` property is preferred because it does not have the overhead of a function call.

Quindi occorre usare `length`. Esempio:

```
alert($('li').length); // 5
```

Funzionalità deprecate in jQuery

In questo articolo vedremo quali metodi e selettori sono stati deprecati in jQuery. Essere deprecato non implica l'abbandono del supporto da parte della libreria, ma più semplicemente l'invito ad utilizzare altre soluzioni.

La pagina ufficiale riporta:

These items have been deprecated, though not necessarily removed, from jQuery. Most of them represent functionality that is unpopular, confusing, inefficient, or ineffective. The specific API entries have advice on alternatives.

Pagina ufficiale

Deprecated

I problemi dell'evento load() di jQuery con le immagini

L'evento `load()` di jQuery è stato deprecato ufficialmente a causa di vari problemi sorti durante l'implementazione e, soprattutto, delle differenti interpretazioni date dai vari browser. Vediamo in dettaglio questo problema.

La documentazione ufficiale riporta ad esempio i seguenti problemi dell'evento `load()` con le immagini:

A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:

- It doesn't work consistently nor reliably cross-browser
- It doesn't fire correctly in WebKit if the image src is set to the same src as before
- It doesn't correctly bubble up the DOM tree
- Can cease to fire for images that already live in the browser's cache

L'evento non funziona in modo cross-browser (specie nei browser basati su WebKit), non attiva correttamente la fase di bubble (risalita) dell'evento nell'albero del DOM e può non funzionare quando le immagini sono presenti nella cache del browser.

Esempio:

```
$('img#book').load(function() {   // ... });
```

Per i motivi sopra elencati se ne sconsiglia l'uso.