

Le librerie jQuery e jQueryUI

Rev Digitale 4.3 del 08/11/2019

jQuery

Il selettore jQuery	2
La gestione degli eventi	3
Elenco dei principali metodi di evento	4
Elenco dei principali metodi jQuery	5
Effetti grafici	6
Richiamo dei metodi in cascata	7
Oggetti jQuery e oggetti javascript	8
L'oggetto event ed il passaggio dei parametri agli eventi	9
Richiamo sugli psudoselettori CSS	11
Navigazione della collezione restituita dal selettore (traversing)	12
Creazione dinamica di nuovi elementi	15
Il metodo <code>_table.DataTable</code>	18
Creazione di nuovi attributi	19
Directly-Bound-Events and Delegated-Events	20

jQueryUI

Introduzione a jQuery User Interface	22
Animazioni sui colori	23
Animazioni sulle classi	23
Il metodo <code>effect()</code>	23
Le classi di interazione	24
Eventi e Proprietà	24
La gestione degli eventi e i parametri <code>events</code> e <code>args</code>	25
La gestione dei metodi	26
La classe <code>resizable</code>	26
La classe <code>draggable</code>	27
La classe <code>droppable</code>	28
La classe <code>selectable</code>	29
La classe <code>sortable</code>	30
I widget	31
<code>button</code>	31
<code>controlgroup</code>	31
<code>autocomplete</code>	32
<code>dialog</code>	32
<code>accordion</code> e <code>tabs</code>	33
<code>datepicker</code>	34
<code>menù</code> e <code>sottomenù</code>	35
<code>progress Bar</code>	36
<code>slider</code>	37
<code>spinner</code>	38
Deferred and Promise	39

jQuery

The Write Less, Do More, JavaScript Library (*jquery.com*). jQuery è una libreria Javascript

- mirata a **semplificare l'interazione con il DOM** della pagina HTML e quindi a **velocizzare la scrittura del codice**
- **cross-browser**, fornisce cioè un livello di astrazione tale da azzerare le differenze tra i browser

Pubblicata per la prima volta il 22 agosto 2005 da John Resig, ha raggiunto la versione 1 (stabile) il **26 agosto 2006**.

Ultime versioni (data rilascio ufficiale)

2.1.4	ottobre 2015
3.2.1	ottobre 2017
3.3.1	gennaio 2018

Collegamento offline

E' possibile scaricare la libreria dal sito **jquery.com** in formato compresso (.min) oppure non compresso (.js). Inserire la libreria nella cartella di lavoro e richiamarla come un normale file JavaScript.

```
<script type="text/javascript" src="jquery.js"> </script>
```

Collegamento online

In alternativa è possibile utilizzare un cosiddetto **CDN** (Content Delivery Network) cioè inserire nella pagina html un collegamento ad uno dei vari server **CDN** che ospitano le librerie jQuery. Ad esempio:

```
<script src="http://code.jquery.com/jquery-3.3.1.min.js"> </script>  
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"
```

Il collegamento online ha il vantaggio che, se si è già visitato un sito che usa jQuery, il browser ha la libreria in cache ed evita di doverla ricaricare. Inoltre non appesantisce il nostro server di hosting.

La versione slim non supporta ajax.

Il Selettore jQuery

Il costrutto fondamentale è costituito dalla funzione dollaro `$()` che è un alias di una funzione base denominata `jQuery()`. jQuery significa JavaScript Query, nel senso che **riceve come parametro, una stringa costituita da un generico selettore CSS, ricerca tutti gli elementi della pagina che soddisfano alle condizioni del selettore CSS e restituisce un vettore enumerativo di puntatori javascript relativi a tutti gli oggetti individuati**, anche nel caso in cui l'oggetto individuato sia **uno solo**.

- `$("p")` Tutti i tag `p` della pagina HTML. Oppure `jQuery("p")`
- `$("p.grassetto")` Tutti i tag `p` della pagina HTML che implementano la classe `grassetto`
- `$("p:nth-of-type(1)")` Tutti i tag `p` che sono i primi del loro tipo
- `$("#wrapper")` Elemento avente `id = "wrapper"`
- `$(".myClass")` Tutti gli elementi che implementano la classe `myClass`
- `$("#wrapper p")` Tutti i tag `p` contenuti all'interno del contenitore avente `id=wrapper`
- `$("#wrapper, p")` Il tag `wrapper` e tutti i tag `p` contenuti nella pagina
- `$("body")` Il tag `body`

E' anche possibile accedere agli oggetti del DOM, nel qual caso occorre omettere le virgolette

- `$(document)` Il documento corrente (evento ready: `$(document).ready(function() {});`)
- `$(window)` la finestra corrente (evento scroll: `$(window).scroll(function() {});`)

La gestione degli Eventi

Per collegare un evento ad una funzione occorre utilizzare il metodo **on** avente due parametri:

- nome dell'evento html da collegare
- funzione **priva di parametri** da eseguire in corrispondenza dell'evento.

```
$("#div#wrapper").on("click", function() {  
    ...codice da eseguire    });
```

La funzione di evento può essere scritta:

- In forma **anonima** direttamente all'interno dell'evento.

```
$("#myButton").on("click", function() {  
    $("#div#wrapper1").hide( );  
});
```

Questa sintassi ha il vantaggio di poter accedere a tutte le variabili esterne appartenenti al contesto in cui trova e presenta anche una maggiore leggibilità.

- In forma **esplicita**, cioè come funzione esterna di cui si passa il riferimento.

```
$("#myButton").on("click", nascondi);    // senza tonde e senza ;  
function nascondi() {  
    $("#div#wrapper1").hide( ); }  

```

nascondi in pratica è una variabile di tipo reference che contiene un riferimento a funzione.

Questa sintassi ha il vantaggio di poter richiamare più volte la funzioni da ambiti differenti

*E' anche possibile assegnare il riferimento a funzione direttamente ad una variabile e poi passare la variabile all'evento. In questo caso però la variabile DEVE essere dichiarata **prima** della procedura che la utilizza. Nei casi precedenti, invece, la procedura nascondi può essere dichiarata indifferentemente **prima o dopo** del suo utilizzo*

```
var nascondi = function () { }  
$("#myButton").on("click", nascondi);
```

Handler multipli

Mediante più on() successivi è possibile collegare due o più handler ad uno stesso evento.

```
$("#div#wrapper").on("click", funzione1);  
$("#div#wrapper").on("click", funzione2);
```

Attenzione però che **se si fa due volte il bind ad uno stesso handler, questo verrà eseguito due volte!** Se si vuole fare il bind ad un evento non sapendo se questo è già stato impostato oppure no, è consigliato fare **prima** l'off (che in caso di evento non associato risulta influente) e **dopo** l'on:

Il metodo off()

Il metodo **off** consente di rimuovere un handler di evento. Comodo per evitare associazioni multiple.

- Richiamato senza parametri, rimuove tutti i gestori di evento associati all'oggetto
`$("#p").off();`
- Richiamato passando il nome di un evento, rimuove tutti i gestori di evento associati a quell'evento
`$("#p").off("click");`
- Se si desidera rimuovere l'associazione ad un singolo evento occorre specificare esplicitamente il nome del gestore da rimuovere;
`$("#p").off("click", nascondi);`

In questo caso il gestore di evento deve necessariamente essere stato espresso in modo esplicito

Il metodo one

Il metodo “one” è simile a “.on” ma l’evento viene eseguito una sola volta per ogni elemento. Nel senso che sul primo click l’handler viene eseguito, sul secondo click sul medesimo elemento l’handler non viene più eseguito. **Non risolve però il problema dei bind multipli, risolvibile solo tramite .off()**

Collegamento diretto di un handler di evento

Per alleggerire la scrittura del codice, è stato introdotto un alias che consente di richiamare direttamente l’evento partendo dal selettore:

```
$("#div#wrapper").click(function() {});
```

Ad esempio l’evento `$(document).ready(function() {})`, richiamato automaticamente al termine del caricamento della pagina, non supporta la sintassi `on` ma soltanto il collegamento diretto.

Il collegamento diretto consente però soltanto la gestione dei cosiddetti “**directly bound events**” e non consente la gestione dei “**delegated events**”. (pag20).

Nota: In alternativa a `on()` e `off()` si possono utilizzare `bind()` e `unbind()` che presentano la stessa sintassi di `on()` e `off()` ma NON consentono l’utilizzo dei delegated events. Sono cioè **equivalenti** al collegamento diretto dell’evento. Esistono anche `delegate()` e `undelegate()` che sono anch’essi analoghi a `on()` e `off()` ma consentono SOLO l’utilizzo dei delegated events. `on()` e `off()` sono i più flessibili.

Generazione forzata di un evento

```
_list.trigger("change");
```

Forza l’evento `change` del listBox senza la selezione di una nuova voce (con la semplice modifica da codice di `selectedIndex` l’evento non viene generato). Utile per eseguire un refresh dopo un elimina

Elenco dei principali metodi di evento

I metodi d evento sono tutti scritti completamente in minuscolo, senza camelCasing.

```
.click() - .dblclick()
.change() //cambio della selezione di un ListBox
           //cambio del contenuto del textBox (richiamato solo quando si abbandona il campo)

.mouseenter() - .mouseleave() // Richiamati quando il cursore entra / esce dall’elemento
.mouseover() - .mouseout() // Analoghi. Vengono però richiamati anche quando il
                           cursore entra / esce su elemento figlio interno all’elemento corrente
.mousemove() // Richiamato durante il movimento del sensore all’interno dell’elemento o del figlio
.hover() // Riceve 2 funzioni di callback equivalenti a .mouseenter() e .mouseleave()

.keypress()
.keyup()
.keydown()

.focusin() // Richiamato nel momento in cui l’elemento riceve il focus
.blur() // Perdita del focus
.scroll() // scroll della pagina o dell’elemento
.resize() - ridimensionamento della pagina.
.submit() - evento associato alla form (e NON al pulsante di submit).
            Per annullare il submit return false; oppure event.preventDefault();
```

Elenco dei principali metodi jQuery

Elenco completo alfabetico di tutti i metodi ed eventi jQuery su: <http://api.jquery.com>

Notare che jQuery sostanzialmente non ha Property ma **ha solo Metodi()**.

<code>.text();</code>	Restituisce il valore di un qualsiasi oggetto testuale restituito dal selettore.
<code>.text("salve");</code>	Assegna un valore ad un qualsiasi oggetto testuale
<code>.html("salve");</code>	Uguale a text, ma all'interno del testo accetta tag HTML (preferibile) E' l'equivalente di innerHTML di javascript.
<code>.html();</code>	Restituisce il valore di un qualsiasi oggetto testuale, compresi eventuali tag html contenuti all'interno dell'elemento. Es: <code>\$("p").text(); // "testo del paragrafo"</code> <code>\$("p").html(); // "testo del paragrafo"</code>
<code>.empty();</code>	Assegna un stringa vuota ad un qualsiasi oggetto testuale. (<code>innerHTML=""</code>)
<code>.val();</code>	Restituisce il value dei controlli restituiti dal selettore corrente
<code>.val("salve");</code>	Assegna un valore all'attributo <code>value</code> di tutti i controlli restituiti dal selettore. E' possibile passare valori multipli racchiudendoli fra quadre: (['uno', 'due'])
<code>.width()</code> e <code>.height()</code>	Consentono di leggere / scrivere direttamente width e height
<code>.css("property-Name")</code>	Restituisce il valore di una singola CSS property del tag corrente
<code>.css("property-Name", "valore")</code>	Consente di assegnare una CSS property al tag corrente La property deve essere scritta sempre CON le virgolette, <u>sia se espressa in formato css</u> (es text-align) <u>sia se espressa in formato camelCasing</u> . I valori numerici possono essere scritti CON apici e UM oppure senza apici e UM

Per le assegnazioni multiple è utilizzabile anche con la seguente comodissima sintassi:

```
var box = { height:"50px", width:"50px"}
$("#myDiv").css(box);
```

In questo caso le chiavi in formato camel-casing possono essere scritte indifferentemente con o senza virgolette. Questa sintassi concatena le property senza sovrascrivere, dunque può essere utilizzata indifferentemente prima / dopo della sintassi precedente.

Una property CSS non assegnata contiene tipicamente il valore **"none"** oppure **stringa vuota**.

<code>.addClass("nomeClasseSenzaPuntino")</code>	Applica una classe CSS all'elemento corrente
<code>.removeClass("nomeClasseSenzaPuntino")</code>	Rimuove una classe CSS dall'elemento corrente E' possibile applicare/rimuovere contemporaneamente più nomi di classi separati da uno spazio
<code>.hasClass("nomeClasseSenzaPuntino")</code>	Restituisce true se l'elemento implementa la classe
<code>.toggleClass("nomeClasseSenzaPuntino")</code>	Aggiunge / Rimuove la classe
<code>.offset().left/top</code>	Consente di leggere / impostare rapidamente left / top senza passare attraverso la proprietà <code>css("propertyName")</code>

I metodi attr() e prop()

Sono molto simili e spesso intercambiabili.

.attr() è utilizzato per accedere agli attributi html veri e propri (es id, name, href) impostati staticamente all'interno della pagina HTML oppure impostati mediante attr stesso.

Accede alle variabili booleane secondo le regole di html5: `.attr("disabled", "disabled");`

.prop() è utilizzato per accedere alle property non definite in html, come ad esempio **selectedIndex**, e per modificare dinamicamente a run time i valori degli attributi veri e propri, in modo da conservare i valori originali all'interno di .attr(). Il valore iniziale di prop() coincide con il valore iniziale di attr().

Accede alle variabili booleane con true/false: `.prop("disabled", true);`

`.attr("AttributeName")` Restituisce il valore di un attributo HTML del tag corrente,
`.attr("AttributeName", "valore")` Imposta un attributo HTML del tag corrente:
`.attr("disabled", "disabled")` Per gli attributi booleani si ripete il nome stesso dell'attributo
`.removeAttr("disabled");` Rimuove un attributo HTML (come se lo cancellassi dal tag)
`.attr({ "attrName": "valore", etc })` Sintassi alternativa per assegnazioni multiple
`$("#myLink").attr({ "href": "http://www.html.it", "target": "_blank" });`
`.prop("AttributeName")` Restituisce il valore di un attributo HTML impostato dinamicam.
`.prop("AttributeName", "valore")` Consente di assegnare un valore ad un attributo HTML. Es:
`.prop("disabled", true/false);` Con **prop** occorre utilizzare true/false.
 E' anche disponibile la seguente sintassi per assegnazioni multiple:
`.prop({disabled:true, target:"_blank"})`

Effetti grafici

`.show()` Applicato ad un elemento nascosto, lo visualizza.
 Il metodo **show** effettua una visualizzazione con **dissolvenza spaziale**, nel senso che l'oggetto viene visualizzato gradatamente partendo dallo spigolo in alto a sinistra e poi, progressivamente, viene mostrato il resto.
`.hide()` Applicato ad un elemento visualizzato, lo nasconde.
 E' l'equivalente di `.css({ "display": "none" });` per cui l'elemento viene rimosso dal normale rendering della pagina con conseguente spostamento degli elementi vicini
 Se ad un elemento interno ad un tag DIV viene applicato il metodo `hide()`, quando il tag DIV verrà visualizzato l'elemento nascosto continuerà a rimanere nascosto
`.fadeIn()` Applicato ad un elemento nascosto, lo visualizza.
 Il metodo **fadeIn** effettua una visualizzazione con **dissolvenza temporale**, nel senso che l'oggetto viene visualizzato tutto insieme ma in modo sgranato e sbiadito e poi poco alla volta l'immagine prende forma e si colora.
`.fadeOut()` Applicato ad un elemento visualizzato, lo nasconde.
`.slideDown()` Applicato ad un elemento nascosto, lo visualizza con una transizione verso il basso.
`.slideUp()` Applicato ad un elemento visualizzato, lo nasconde con una transizione verso l'alto.
`.toggle()` Se l'elemento è visualizzato esegue `hide()`, altrimenti esegue `show()`.
 Ovviamente va utilizzato su elemento diverso rispetto a quello su cui si fa il click, altrimenti quando l'elemento viene nascosto non si può rifare il click(),
`.slideToggle()` Uguaile a `toggle()` però riferito a `slideDown()` e `slideUp()`

In realtà tutti i metodi precedenti (compreso `toggle`) presentano alcuni parametri facoltativi :

- 1) **duration** che indica la durata dell'animazione. Si possono utilizzare i valori "fast" (200 ms) e "slow" (600 ms) oppure un tempo espresso in msec. Se non si specifica la duration :
 - Nel caso di `show()`, `hide()` e `toggle()` il default è 0
 - Nel caso di `fadeIn()`, `fadeOut()` e `slideDown()` e `slideUp()` il default è 400ms
- 2) **ease** che può assumere solo 2 valori: "swing" (default) e "linear". Valori aggiuntivi sono disponibili utilizzando jqueryUI o altri plugin aggiuntivi
- 3) **complete** (ultimo parametro) indica una funzione di callback da eseguire al termine dell'animazione. Gli eventuali stili impostati dalla funzione di callback rimarranno attivi anche al termine dell'animazione, fino a quando non verranno esplicitamente rimossi / modificati.

```
.slideUp(1500, callback); // oppure
.slideUp({ duration: "slow", complete: callback});
```

Il metodo animate()

```
.animate ( {width:"100px", etc}, mSec, "linear", callback);  
.animate ( {left:"+=600px"}, 3000, callback);
```

Applica una animazione alle proprietà indicate dal primo parametro, intese come variazione rispetto al loro valore corrente. Il secondo parametro indicata la durata dell'animazione.

Il terzo parametro rappresenta una funzione di **callback** che, esattamente come nei casi precedenti, verrà richiamata soltanto al termine dell'animazione.

Consente di animare qualsiasi proprietà CSS che abbia un valore numerico, quindi ad esempio margini, padding, bordi, altezza, larghezza, posizioni, opacità.

Es: `$(ref).animate({opacity:0},1000);`

NON consente invece l'animazione dei colori, che sarà possibile solo con le librerie UI.

Gli stili impostati rimangono attivi anche al termine dell'animazione, fino a esplicita rimozione.

Thread separati

Tutti gli effetti animati vengono avviati all'interno di un thread separato, per cui scrivere:

```
$("#myDiv").fadeIn(1000);  
$("#myDiv").text("salve mondo");
```

nell'ordine indicato oppure in ordine inverso è assolutamente la stessa cosa. In entrambi i casi il `.text` viene applicato subito, mentre il `fadeIn` visualizza lentamente il tag DIV (inizialmente nascosto).

Notare che animazioni successive su uno stesso oggetto vengono **accodate**, per cui qualsiasi ulteriore animazione verrà eseguita soltanto **al termine** dell'animazione precedente.

Il metodo delay()

Il metodo **delay(msec)** richiamato fra due animazioni successive, consente di introdurre un ritardo fra le due animazioni. **Non è utilizzabile per ritardare l'applicazione di una proprietà all'interno del thread principale**, ma solo sui thread secondari relativi alle animazioni. Es

```
$("#mydiv").fadeIn(500).delay(500).fadeOut(500); // ok  
$("#myDiv").delay(1500).css("background-color", "#eee") // nok viene applicata subito
```

Il metodo stop()

Il metodo `.stop()` consente di terminare l'animazione in corso sull'oggetto corrente.

Il metodo `.stop(true)` consente di svuotare la code delle animazioni pendenti sull'oggetto corrente

Il metodo .disableSelection();

E' un metodo di **jQueryUI** che, applicato ad un **wrapper**, inibisce la selezione del testo negli elementi interni rispetto al contenitore corrente. E' analogo alla property CSS3 **user-select:none**

Richiamo dei metodi in cascata

Quasi tutti i metodi jQuery restituiscono il puntatore `this` all'oggetto stesso, per cui è sempre possibile richiamare i metodi in cascata, uno in sequenza all'altro:

```
$("#myTag").hide().slideDown("slow");  
Il tag viene dapprima nascosto, e poi ri-visualizzato con velocità lenta.  
$("#div.lampadina").addClass("accesa").fadeIn(2000);
```

La lampadina, che è nascosta, viene prima colorata di giallo, e poi mostrata in un tempo di 2 sec.

Anche i **Metodi di evento** restituiscono un puntatore all'elemento, per cui sono anch'essi concatenabili

```
$("div#wrapper").show( )
    .mouseover( function() { $("#desc1").fadeIn ( ); } ) // evento
    .mouseout ( function() { $("#desc1").fadeOut ( ); } ); // evento
```

La collezione jQuery ed i puntatori JavaScript

- Il selettore \$ restituisce **sempre** una **Collezione di oggetti del DOM** da intendersi sostanzialmente come un **Vettore Enumerativo di puntatori javascript**.
- La proprietà `.length` restituisce il numero di elementi appartenenti alla collezione.
- **Il test su `.length==0` rappresenta il modo migliore per testare se la collezione è vuota** (il test del puntatore su null o undefined non funziona in quanto il vettore esiste, ma la sua lunghezza è zero)
- Il vettore enumerativo sempre vettore rimane anche quando l'elemento restituito è uno solo, nel qual caso sarà un vettore con lunghezza 1.
- **I metodi jQuery, in scrittura, applicano il loro effetto a tutti gli elementi della collezione,**
- **I metodi jQuery, in lettura, applicano il loro effetto soltanto al 1° elemento della collezione.**

Si consideri il seguente esempio:

```
var _div = $("div"); // collezione di elementi div
alert(_div.html()); // valore del primo elemento
_div.css({backgroundColor:"red"}) // applicato a tutti gli elementi
_div[2].innerHTML="Sono il terzo div della collezione jQuery";
```

Il metodo `html()` restituisce il valore del primo elemento della collezione, mentre il metodo `.css()` applica l'effetto a tutti gli elementi della collezione. L'ultima riga accede al 3° elemento della collezione (a base 0) inteso come puntatore javascript e va a modificarne il contenuto

Il selettore jQuery accetta come parametro anche un **puntatore javascript** nel qual caso restituisce **sempre una collezione** con all'interno quell'unico puntatore. Esegue una specie di cast.

```
var ref = document.getElementById("myDiv");
alert( ref.innerHTML);
alert( $(ref).html());
```

In base a questa considerazione, l'ultimo esempio iniziale potrebbe essere scritto nel modo seguente

```
$ ( $( "div" ) [ 2 ] ).html("Sono il terzo div della collezione jQuery");
```

Il fatto che i metodi jQuery in scrittura applichino il loro effetto a tutti gli elementi di una collezione, può essere sfruttato anche per **l'associazione degli eventi**:

```
$("input[name=opt]").on("click", function() {});
```

L'evento click viene assegnato a TUTTI i radio buttons; all'interno dell'evento occorre però accedere selettivamente a quello selezionato: `let _opt= $(this)` oppure `$("input[name=opt]:checked")`

Aggiunta di nuovi elementi ad una collezione

Tramite il metodo **push()** è possibile aggiungere nuovi elementi alla collezione restituita da un selettore:

```
var collection = $(".myClass");
collection.push(_carta); // analogo a _carta.addClass("myClass");
```


Il confronto fra puntatori

Non è consentito eseguire il confronto fra due collezioni jQuery. Se si tenta di eseguire il confronto fra due puntatori Query, anche se questi puntano alla stessa identica collezione o puntano allo stesso identico elemento, questo confronto ritorna sempre un esito negativo in quanto il confronto viene fatto sui puntatori ai vettori che, ovviamente sono differenti.

```
var _div1 = $("#myDiv"); // id univoco
alert(_div1.text());
var _div2 = $("#myDiv");
alert(_div1.text());

if(_div1==_div2) // sono due collezioni differenti
    alert("impossibile");
if(_div1[0]==_div2[0])
    alert("OK! i due puntatori puntano allo stesso oggetto!");
```

L'ultima riga accede all'elemento 0 di ciascuna collezione ed esegue correttamente il confronto

L'oggetto event

In jQuery tutte le procedure di evento ricevono in realtà un parametro (facoltativo) denominato **event** che contiene alcune informazioni relative all'evento generato. Riguardo a questo parametro, i vari browser utilizzano di solito sintassi differenti per cui, da punto di vista, jQuery ha il pregio di uniformare le sintassi. Ogni procedura di evento riceve sempre un oggetto event indipendentemente dal browser in uso.

Questo oggetto contiene diverse property fra cui:

event.type	Il nome dell'evento
event.timeStamp	tempo trascorso (in millisecondi) dal caricamento della pagina
event.target	puntatore js all'elemento che ha scatenato l'evento. Analogo a this
event.currentTarget	puntatore js all'elemento attuale a cui è collegato l'evento
event.keyCode	nel caso di tastiera contiene il codice fisico del tasto premuto
event.which	nel caso di tastiera e mouse contiene informazioni sul tasto premuto
event.data	oggetto contenente eventuali parametri aggiuntivi

Gli eventi si propagano per default in bubbling phase verso gli elementi ascendenti del DOM che, dentro target, vedono l'elemento che ha scatenato l'evento mentre dentro currentTarget vedono se stessi.

Passaggio dei parametri ad un gestore di evento

Il terzo parametro del metodo on consente di passare dei parametri alla procedura di evento che li potrà leggere all'interno del campo data. Questi parametri possono essere passati in formato json come object oppure come oggetto serializzato (stringa). Se si passano come stringa occorre passare anche il secondo parametro "selector", eventualmente impostandolo al valore "". Es

```
$("button").on("click", {name: "Karl"}, visualizza );
$("button").on("click", "", {"name":"Karl"}, visualizza );

function visualizza(event) {
    alert( "Hello " + event.data.name ); }
```

Note

Impostazione dell'ID

Se l'ID di elemento HTML deve poi essere utilizzato all'interno di un selettore jQuery, **NON** deve contenere la **virgola**, perché verrebbe interpretata come separatore di classi CSS.

Stesso problema con il **puntino** e con i **due punti**. Vanno bene invece trattino e underscore.

Alias dell'evento `$(document).ready(function() { })`

Il selettore `$` può essere applicato anche direttamente ad una funzione, nel qual caso diventa sostanzialmente un alias dell'evento `$(document).ready()`. La sintassi è la seguente:

```
$(function() { } )
```

Puntatore javascript diretto

Nel caso dei selettori assoluti (es `#wrapper`), da jQuery 1.9 in avanti viene creata automaticamente anche una variabile avente il nome dell'ID che rappresenta un **puntatore java script** all'elemento in questione. Antepoendo il dollaro si ottiene il corrispondente puntatore jQuery:

```
$(wrapper).css("background-color", "red");
```

this

Il puntatore `this` rappresenta sempre il puntatore all'oggetto che ha avviato la funzione di callback, sia nel caso delle funzioni scritte in loco, sia nel caso delle funzioni di callback scritte esternamente.

Passaggio dei parametri ad una funzione di callback

Per quanto riguarda invece i parametri, se il metodo si aspetta come parametro una funzione di callback che a sua volta riceverà dei parametri al momento dell'invocazione,

- nel caso di funzione di callback scritta in forma anonima all'interno del metodo, tali parametri devono essere indicati esplicitamente all'interno della firma stessa
`http.createServer(function (richiesta, risposta) { }`
- nel caso invece di funzione di callback scritta esternamente, questa dovrà essere passata al metodo priva di parametri, in quanto si passa in pratica il **puntatore** alla funzione:
`http.createServer(rispondi) { }`

La funzione scritta esternamente dovrà però avere la firma appropriata:

```
function rispondi (request, response){
```

Quasi tutti i metodi jQuery richiedono funzioni di callback prive di parametri.

Richiamo sugli PseudoSelettori CSS

Sono introdotti dal simbolo **:**

```
:checked :enabled :disabled
```

possono essere utilizzati all'interno di una qualunque espressione CSS.

```
:selected :visible :hidden :button :radio :submit
```

sono disponibili soltanto tramite selettore jQuery e non tramite CSS.

La seguente sintassi con parentesi quadra vale per **molte** attributi html ma non per tutti:

```
input[type=radio]
input[type=checkbox]
input[type=text]
input[type=button]
input[type=submit]
```

```
input[name=txtNome]
input[name=opt1]
input[name=chk]:checked
p[style="font-style:italic"]
```

In questo caso il confronto viene fatto con l'**INTERO** attributo **style** impostato dinamicamente

Esempi

```

$:disabled").prop("disabled", false);
alert( $("input[name=chk]:checked").val() ) // value del 1° checkbox checked
alert( $("#1stElenco option:selected").val() )
$("select option[value=3]").prop('selected','selected');

```

CSS Functions

```

:not(selettore_Secondario) // Es input[type=radio]:not(:checked)
:contains(testo);         // TRUE se l'elemento contiene il testo indicato (anche annidato)
#menu li:has(ul)          // Le voci di menu che contengono un tag UL (anche annidato)

```

PseudoClassi di filtro su un gruppo di elementi

```

:first-child           // Primo figlio generico (indipendentemente dal tipo)
:first-of-type         // Primo elemento del suo tipo

```

:first-child restituisce true se l'elemento corrente gode della proprietà di essere primo figlio (primogenito) del proprio genitore, qualunque sia il genitore, e indipendentemente dal proprio tipo.

```

<div id="wrapper">
  <p> ..... </p>
  <p> <input .....> </p>
  <input .....>
</div>

```

Il primo <input> presenta :first-child uguale a true perché è in assoluto il primo figlio di <p>
 Il secondo <input> presenta :first-child uguale a false perché dentro wrapper, prima di lui, ci sono altri figli. Il primo <input> presenta :first-of-type uguale a true perché, rispetto al proprio padre, è il primo elemento di tipo input. Il secondo <input> presenta :first-of-type uguale a true perché, all'interno di wrapper, è il primo elemento di tipo input

```

:last-child           // Ultimo figlio generico (indipendentemente dal tipo)
:last-of-type         // Ultimo elemento del suo tipo

:nth-child(i)         // i-esimo figlio generico (indipendentemente dal tipo) (a base 1)
:nth-of-type(i)       // i-esimo elemento del suo tipo (a base 1)

div :nth-child(i)     // i-esimo elemento generico contenuto in un tag div
div :nth-of-type(i)   // i-esimo elemento del suo tipo contenuto in un tag div
:only-child           // Figlio unico

div p:nth-child(i)    // i-esimo figlio generico di div, se di tipo p

```

In quest'ultimo caso occorre prestare molta attenzione. Se nth-child(i) all'interno di div non è di tipo p, il selettore restituisce false. Vale anche per le classi:

```

div .myClass:nth-child(i) // i-esimo figlio generico di div, se implementa la classe myClass
div .myClass:nth-of-type(i) // i-esimo elemento del suo tipo, se implementa myClass

```

```

<div id="titolo"> titolo </div>
<div class="riga"> 1 </div>    <-- #wrapper .riga:nth-of-type(2)
<div class="riga"> 2 </div>    <-- #wrapper .riga:nth-of-type(3)

```

Gli pseudoselettori :nth-child() e :nth-of-type() oltre all'indice i-esimo accettano come parametro anche i valori: **even** tutti gli elementi pari - **odd** tutti gli elementi dispari
even e **odd** sono utilizzabili sempre solo in abbinamento con nth-child() e nth-of-type()

Gli pseudoselettori **:first** e **:last** possono essere accodati a qualsiasi elemento per individuare rispettivamente il primo e l'ultimo elemento della collezione. Possono essere ulteriormente accodati anche a **nth-child(even)** e **nth-of-type(odd)**.

Sono analoghi ai metodi jQuery **.first()** e **.last()**

Metodi di traversing

.length (property) restituisce il numero di elementi individuati dal selettore.

Il metodo **.is("selettoreSecondario")** accetta come parametro un selettore secondario.

Va usato tipicamente all'interno di una IF per vedere **se l'elemento corrente appartiene ad una certa collezione, quella identificata dal selettore secondario**.

Se l'elemento appartiene alla collezione identificata dal selettore secondario restituisce **true**, altrimenti restituisce **false**. Se la collezione secondaria è vuota restituisce sempre **false**.

```
if (_current.is('.myClass'))
if (_lampadina.is('.accesa'))
if ($("#myChkBox").is(':checked'))
```

.filter(sel)

Consente di applicare dei filtri sulla collezione restituita dal selettore jQuery principale. Fra gli elementi restituiti dal selettore principale vengono mantenuti soltanto quelli che soddisfano il selettore secondario. All'interno del selettore secondario è possibile utilizzare qualsiasi selettore / pseudoselettore CSS.

```
$("#li").filter(":nth-child(even)")

$("#contenitore a").filter(".myclass"); // equivale a:
$("#contenitore a.myclass");           // equivale a:
$("#contenitore a:filter(.myclass)) ;
```

.not(sel)

E' l'inverso del precedente. Restituisce solo quegli elementi che NON soddisfano il selettore secondario.

```
$("#contenitore a").not(".myclass"); // equivale a:
$("#contenitore a:not(.myclass)) ;
```

.get(i) consente di accedere ai singoli puntatori JavaScript contenuti in una collezione jQuery.

E' sostanzialmente analogo ad utilizzare [] in coda al nome del selettore.

.eq(i) consente di accedere ai singoli elementi contenuti in una collezione jQuery, restituendoli però sotto forma di **oggetti jQuery**. Entrambi accettano come parametro un indice numerico

Passando come parametro un indice numerico, restituisce il puntatore all'elemento corrispondente.

```
var elenco = $("p").eq(); // Senza parametri non è significativo e può essere omesso
alert (elenco.length);
var ref = $("p").get(0); // puntatore java script
ref.style.backgroundColor="blue";
var ref = $("p").eq(1); // puntatore jQuery
ref.css("background-color", "red");
```

Attenzione che questi metodi, a differenza delle CSS precedenti, sono tutti a base 0 !

.slice(i, j) simile a **.eq()** ma restituisce una collezione di elementi, il primo compreso, l'ultimo escluso

.each() consente di scorrere tutti gli elementi restituiti dal selettore.

E' sostanzialmente equivalente ad un ciclo for tradizionale avente all'interno una eq().

Come parametro si aspetta una function di elaborazione di ogni singolo elemento scandito.

```
$( "p" ).each(function () {  
    alert( $(this).text() );  
});
```

La funzione passata al metodo **.each()** in realtà riceve automaticamente due parametri che sono:

l'**indice** dell'elemento corrente ed un riferimento all'elemento stesso (sotto forma di riferimento javascript

```
$(selettore).each(function (i, ref) {  
    alert( $(ref).html() ); // E' anche possibile indicare solo il primo parametro i,
```

I metodi each() sono sempre sincroni (bloccanti), nel senso che interrompono l'esecuzione fino a quando non sono terminati. Il metodo statico **\$.each()** serve per scandire un semplice vettore.

.children("selettoreSecondario") restituisce la collezione dei figli dell'elemento corrente che soddisfano al selettore secondario impostato. La proprietà **.length** restituisce il numero di elementi. Come parametro accetta soltanto un **selettoreSecondario** che consente di restringere la collezione degli elementi individuati. **Non sono ammessi indici numerici.**

Ad esempio **.children("p")** restituisce tutti i figli di tipo "p".

Attenzione che I figli individuati da **children** sono **SOLO quelli di primo livello (figli diretti)**;

```
$("#wrapper").children("div") equivale a $("#wrapper > div")  
    //primo figlio di wrapper  
$("#wrapper").children(":first-child").css("font-style") = "italic";  
    //ultimo figlio di wrapper di tipo div  
$("#wrapper").children("div:last-child")
```

Se last-child non è di tipo DIV viene restituita collezione vuota

.find(sel) simile a children() ma restituisce **figli e nipoti**

.parent()

Il metodo parent() opera al contrario di children(), cioè restituisce i genitori **diretti** di tutti gli elementi individuati dal selettore. Ammette come parametro un selettore secondario di filtro

.index(selector) è l'inverso di eq(). Restituisce l'indice dell' elemento individuato dal selettore interno (parametro) rispetto alla collezione individuata dal selettore principale. Restituisce -1 se l'elemento non viene trovato. Il parametro può essere un **oggetto javascript** oppure un **oggetto jQuery**.

```
var indice = $("p").index($("#xx") ); // oppure :  
var indice = $("p").index(document.getElementById("xx"));
```

Se non si passa nessun parametro, index() restituisce il numero di elementi presenti nel selettore principale. Diventa sostanzialmente uguale a length.

.first() .last()

.first() seleziona il primo elemento della collezione corrente. Analogo a **:first**

.last() seleziona l'ultimo elemento della collezione corrente. Analogo a **:last**

.add(sel)

Consente di aggiungere un ulteriore collezione a quella già individuato dal selettore principale. E' sostanzialmente analogo all'utilizzo della virgola all'interno del selettore CSS.

`.siblings()` restituisce la collezione di TUTTI i fratelli del nodo corrente (nodo corrente escluso).

`.next()` `.nextAll()` `.prev()` `.prevAll()`

`.next()` accede all'elemento successivo (prossimo fratello) rispetto all'elemento corrente (sullo stesso livello di gerarchia).

`.nextAll()` seleziona tutti gli elementi successivi

`.prev()` e `.prevAll()` analoghi. Accede all'elemento/i precedente/i rispetto all'elemento corrente.

```
$("li.corrente").next().css("background-color", "red");
```

`.contents(sel)`

è simile a `children` e restituisce SOLO i figli di primo livello. La differenza sta nel fatto che, rispetto a `children` restituisce anche eventuali nodi testuali (contenuto del tag) ed eventuali comment node. Inoltre consente di accedere anche ai contenuti di un `IFrame`.

Passaggio di una funzione come parametro

Un vantaggio nell'utilizzo dei metodi jQuery rispetto agli pseudoselettori CSS è che risulta possibile passare come parametro una funzione che operi da filtro restituendo true o false.

```
$("#contenitore a").filter(function () {
    return $(this).hasClass("myClass");
});
```

Note operative

Nota 1: Notare la differenza tra le seguenti righe di codice:

```
$("div").text(Math.floor(10*Math.random())+1);

$("div").each(function() {
    $(this).text(Math.floor(10*Math.random())+1);
});
```

La **prima soluzione** assegna lo stesso numero a **TUTTI** gli elementi della collezione.

La **seconda soluzione** assegna invece un numero specifico a ciascun elemento.

`each()` si usa quando occorre fare delle azioni specifiche su ogni singolo elemento del gruppo.

Se si desidera applicare una stessa azione a tutti gli elementi del gruppo, è sufficiente applicare l'azione direttamente sul selettore principale.

Nota 2: Il metodo `.val()` nel caso di ListBox, Option Button e Radio Button

in lettura

Nel caso di select a selezione singola restituisce automaticamente il **value della voce selezionata**.

Se il value della voce selezionata è vuoto restituisce automaticamente il testo html della voce selezionata

Nel caso di select a selezione multipla restituisce un vett enumerativo con i **value di tutte le voci selected**

Nel caso di radio button e check box restituisce il **value della prima voce** (indipendentemente dal fatto che sia selezionata oppure no).

in scrittura

Nel caso del ListBox a selezione singola provoca la selezione dell'elemento indicato.

Nel caso del ListBox a selezione multipla, radioButton e checkBox, occorre necessariamente passare al metodo `val()` un **vettore enumerativo di elementi** e, in tal caso, provoca la selezione di tutti gli elementi aventi il **value** indicato e la deselegione automatica delle altre voci.

```
$(':checkbox').val(['chk-1', 'chk-2']);
```

La selezione di un nuovo elemento da codice NON provoca la generazione dell'evento `change`, che deve essere invocato manualmente tramite il metodo `.trigger()`

Se si rimuove una `OPTION` da un `ListBox`, viene automaticamente selezionata la prima voce della lista

Nota 3: let e var

Si supponga di dover assegnare un evento click ad un serie di elementi `DIV` aventi `id = div1, div2, etc`

```
for (var i=0; i<DIM; i++) {  
    var _div = $("#div" + i);  
    div.prop("codice", i);  
    _div.on("click", function () {  
        elabora(i);  
        elabora($(this).prop("codice"));    });  
    $("#wrapper").append(div);  
}
```

- Se la variabile `i` viene dichiarata come **var**, la variabile viene allocata globalmente per cui, al momento dell'esecuzione dell' `.on("click")` verrà letto ed iniettato il valore di `i` in quel momento, cioè `DIM` (valore assunto dalla variabile `i` al termine del ciclo).
- Prima di ES6 si avviava a questo inconveniente salvando il valore di `i` all'interno di un apposito attributo (ad esempio `codice`) come indicato in blu nel codice precedente
- L'operatore **let**, rispetto a **var**, definisce la variabile a livello locale, deallocandola automaticamente al termine del proprio ciclo di vita. Pertanto, al momento del click, la variabile non esisterà più. Non potendo accedere alla variabile al momento del click, l'interprete, al momento della chiamata, inietta automaticamente il valore corrente di `i` per cui, in questo caso, il click funzionerà correttamente.
- Quanto detto vale però **soltanto** se la variabile dichiarata con `let` non sarà più esistente al momento del click. **In caso contrario** (ad esempio per variabili dichiarate con `let` subito all'inizio del `document.ready()`) verrà iniettato il valore assunto dalla variabile in quel momento (`DIM`).

Nota 4: il ciclo .each()

Se il `bind` viene eseguito all'interno di un ciclo `each`, il valore viene **iniettato al momento della chiamata**, per cui in questo caso `i` assumerà il valore corretto, diverso per ogni elemento.

Viceversa le altre eventuali variabili verranno create separatamente per ogni singola esecuzione:

```
$("#wrapper div").each(function (i, ref)){  
    var cont=0;  
    $(ref).on("click", function(){  
        cont++;  
        alert(cont);    // 1 per tutti  
        alert(i);        // 0, 1, 2, etc  
    });  
}
```

Creazione dinamica di nuovi elementi

```
var ref = $("

</div>");    // </div> può essere omissa


```

Il vantaggio di usare anche il `</tag>` è che all'interno si può inserire qualsiasi sequenza html
`$(" ");`

Notare come `$("div")` accede agli elementi di tipo `div`
mentre `$("<div>")` crea un nuovo elemento `div`

append(ref): Analogo a appendChild di javascript

```
$("#contenitore").append(ref);
```

Accetta come parametro - un riferimento jQuery (esistente o creato ex novo)
 - un **vettore enumerativo** di riferimenti jQuery
 - una stringa html, che viene convertita e poi "appesa".
 In quest'ultimo caso .append() equivale a .html()

appendTo(parent): Sintassi opposta rispetto alla precedente. Ricevuto un nodo lo appende al genitore

```
var box = $("

Elemento <strong>nidificato</strong></div>");
box.addClass("nuovaClasse");
box.css({"background-color":"red"});
box.appendTo(document.body);
box.appendTo("#contenitore")


```

In tutti questi metodi, se l'elemento `ref` appartiene al DOM, viene **tagliato** dal DOM ed incollato nella nuova posizione.

Passaggio dei parametri in fase di creazione

E' anche possibile passare al 'costruttore' eventuali **attributi HTML** e/o **metodi jQuery**.

- Gli **attributi HTML** vengono applicati così come sono scritti
- I **metodi jQuery** vengono **eseguiti** e ricevono come param ciò che viene indicato dopo i 2 punti

```
$("#<p>", {
  "css": { // metodo jQuery
    "background-color": "#ddd",
    "position": "absolute"
  },
  "text": "hello", // metodo jQuery
  "appendTo": "_contenitore",
  "append": [ // le parentesi quadre consentono di appendere più elementi allo stesso tempo
    $("#<label>", { "text": "hobbies" } ),
    $("#<input>", {
      "type": "radio",
      "name": "sports",
      "value": "true"
    })
  ]
})
```

Questa sintassi è applicabile SOLO su tag scritti senza attributi html diretti.

NB: Non è possibile utilizzare una sintassi mista del tipo :

```
let _tr = $("#<tr>", {
  append: [$("#<td>").addClass("td").html(cont), ..... ] })
```

Se si usa questa sintassi **tutti** gli eventuali attributi **devono** essere scritti nel costruttore

```
append: [$("#<td>", {"addClass": "td", "html": cont++}), ..... ] })
```

Aggiunta di nuovi elementi ad un listBox tramite new Option()

Per creare nuovi elementi è anche disponibile la seguente sintassi java script :

```
lst.append(new Option(text, value));
```

Associazione degli eventi agli oggetti creati dinamicamente

A differenza delle proprietà CSS definite all'interno di un foglio di stile che vengono applicate anche a posteriori sui nuovi elementi creati dinamicamente, **l'associazione degli eventi ad un elemento di una collezione jQuery deve essere fatta soltanto DOPO che l'oggetto è stato appeso al DOM.**

Se l'elemento viene rimosso dal DOM e riappeso, l'associazione con l'evento va persa !!

Le seguenti righe posizionate all'interno del document.ready agiscono SOLO per gli elementi esistenti

```
$("#button").on("click", function() {})  
$("#btn1").on("click", function() {})
```

Il selettore jQuery ricerca gli oggetti indicati all'interno del DOM, per cui le righe precedenti funzionano **soltanto** se il button è già stato creato ed appeso al DOM.

Le soluzioni possono essere:

1. appendere l'elemento al DOM prima di assegnare l'evento. **Non sempre è possibile.** Spesso gli elementi vengono appesi al DOM soltanto in corrispondenza di una risposta Ajax per cui i loro eventi non possono certo essere definiti al momento del document.ready.
2. Nel caso di creazione dinamica degli oggetti all'interno di una risposta Ajax, è possibile associare l'evento attraverso la variabile stessa usata in fase di creazione dinamica.

```
btn.on("click", function() {})
```

Soluzione rischiosa. Se la chiamata Ajax esterna viene eseguita più volte, gli eventi associati internamente vengono riassociati più volte. In questo caso prima di associare un evento è bene eseguire sempre il .off() dell'evento medesimo:

```
btn.off().on("click", function() {})
```

Inoltre se l'elemento viene rimosso e riappeso al DOM l'associazione viene comunque persa

3. utilizzare i **delegated Events** (pag20) che è probabilmente **la soluzione migliore.** L'evento viene associato ad un elemento esistente e, nel momento in cui si verifica, viene propagato a tutti gli elementi interni in quel momento esistenti e anche a quelli creati dinamicamente in un secondo tempo. Diventa però più complicato diassociare un evento.

Altri metodi di creazione dinamica

after(ref): aggiunge l'elemento **ref** ricevuto come parametro **dopo** l'elemento restituito dal selettore principale, del quale ne diventa il fratello successivo

before(ref): aggiunge l'elemento **ref** ricevuto come parametro **prima** dell'elemento restituito dal selettore principale, del quale ne diventa il fratello antecedente

insertAfter(origin) e **insertBefore(origin)** sono uguali opposti rispetto ai due precedenti. Partono dall'elemento da inserire e lo incollano prima o dopo dell'elemento **origin** ricevuto come parametro.

Nel caso in cui il selettore principale restituisca più elementi, l'oggetto ricevuto come parametro viene replicato prima/dopo di ogni elemento restituito dal selettore principale. Esempio :

```
<div class="category"> Calcio </div>  
<div class="category"> Nuoto </div>  
$(".category").before("<h3> Sottocategoria </h3>");
```

produce il seguente risultato :

```
<h3> Sottocategoria </h3>  
<div class=" category ">Calcio</div>  
<h3> Sottocategoria </h3>  
<div class=" category ">Nuoto</div>
```

clone(): esegue un duplicato dell'elemento corrente

```
var _div2 = _div1.clone();
```

remove(): rimuove l'elemento dal DOM. Accetta come secondo parametro un sottoselettore che agisce da filtro sul gruppo restituito dal selettore principale:

```
$(".testo").remove(); // rimuove tutti gli elementi con classe .testo
$("div").remove(".testo"); // rimuove tutti i div che implementano la classe .testo
```

wrap(): avvolge ogni singolo elemento individuato dal selettore con il nuovo tag

Notare che **before** e **after** aggiungono dei fratelli prima e dopo, mentre **wrap** "avvolge" con un parent

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

wrapAll(): raggruppa tutti gli elementi individuati dal selettore e li avvolge con un unico nuovo tag

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

wrapInner(): `$("p").wrapInner("");`

```
prima : <p> Testo </p>
dopo  : <p> <span> Testo </span> </p>
```

L'oggetto DataTable()

Si tratta di un plug-in che consente di aggiungere ad una table html tutte le principali funzionalità di base.

<https://cdn.datatables.net/1.10.19/css/jquery.dataTables.css>

<https://cdn.datatables.net/1.10.19/js/jquery.dataTables.js>

Accetta diversi parametri, i principali sono i seguenti (che per default sono tutti TRUE):

```
$('#table_id').DataTable( {
  "bPaginate": true, // paginazione dei record da visualizzare
  "bLengthChange": true, // n. di record per pagina
  "bFilter": true, // ricerca della voce impostata
  "bSort": true, // ordinamento dei record sul click on the header
});
```

Molto interessante è la possibilità di **caricare i dati attraverso chiamate Ajax**.

In questo modo i dati non devono essere tutti presenti in pagina ma vengono richiamati man mano che l'utente scorre le pagine della tabella

```
$('#table_id').DataTable({
  "sAjaxSource": "scripts/server_processing.php"
  "bProcessing": true,
  "bServerSide": true
});
```

Creazione di nuovi attributi

Il metodo **.attr()** consente anche di creare nuovi attributi all'elemento corrente, esattamente come in Java Script. Utile per memorizzare informazioni relative all'elemento. La stessa cosa può essere fatta attraverso il metodo **.prop()** oppure **.data()** che consente di raggruppare tutti i dati dell'oggetto, senza mescolarli con gli attributi / proprietà HTML.

```
myDiv.attr("somma", 0);
myDiv.prop("somma", 0);
myDiv.data("somma", 0);
```

A differenza degli altri due, **all'interno di `data` è possibile anche salvare un puntatore ad object.**

```
$("#div").data("_this", $(this));
```

I metodi `.css()` e `.addClass()` consentono di definire nuove proprietà CSS e nuove classi, eventualmente anche non definiti all'interno dei css:

```
.css("blocked", true)
.addClass("blocked")
```

NOTA sulle classi e selettori

Alcuni metodi CSS (e soprattutto CSS UI) possono avere come parametro una **className** (indicata nella documentazione come **String**), mentre altri metodi possono avere come parametro un **Selettore CSS** (indicato nella documentazione come **Selector**).

- Nel caso del **className** occorre passare come parametro il nome della classe **scritto senza il puntino**. Rientrano in questo caso tutti quei metodi che contengono la parola Class, come ad esempio `addClass`, `removeClass`... Però, ad esempio, anche la proprietà `placeholder` del metodo `Sortable` rientra in quest'ambito e si aspetta un `className`.
- Nel caso del **css Selector** occorre passare come parametro **sempre una stringa** ma questa volta contenente il nome della classe scritto mediante le regole dei CSS, quindi **scritta con il puntino**. Rientrano in questo caso la maggior parte dei metodi jQuery UI.

Metodi Statici della classe jQuery

\$.isArray(obj)

Restituisce TRUE se l'elemento ricevuto come parametro è di tipo Array

\$.inArray(val, vect)

Restituisce la posizione di `val` all'interno del vettore `vect` (a base 0). -1 se non esiste.

\$.grep(vect, function (val) {return val > 0});

Restituisce un nuovo vettore contenente soltanto gli elementi che soddisfano il criterio intero (`val>0`).

\$.each(vect, function (index, item) { ... });

Scandisce tutti gli elementi di un vettore. `index` è l'indice a base 0, `item` il contenuto. Es:

```
$("#select[name='lst']").append(new Option("id"+cnt, item)); cnt++;
// Il 1° parametro è ciò che viene caricato nella proprietà VALUE
// il 2° parametro è ciò che viene visualizzato all'interno della option
```

```
});
```

\$.getScript("module", callback)

Consente di importare un file javascript all'interno di un altro file javascript.

Da utilizzare subito all'interno dell'evento `$(document).ready()`

Si basa sulla funzione javascript ***import*** che però è molto più macchinosa.

```
$.getScript("moment.js", function() {
    alert(moment());
});
```

\$.fn è un metodo che consente di aggiungere nuove funzionalità all'interno di jQuery.

```
$.fn.blueBorder = function(){
    this.each(function(){
        $(this).css("border","solid blue 2px");
    });
    return this;
};
$('.myTags').blueBorder();
```

Direct Bound Events and Delegated Events

<http://api.jquery.com/on/>

Il metodo `.on` presenta in realtà una firma completa costituita da ben 4 parametri:

`.on(event [, selector] [, data], handler)`

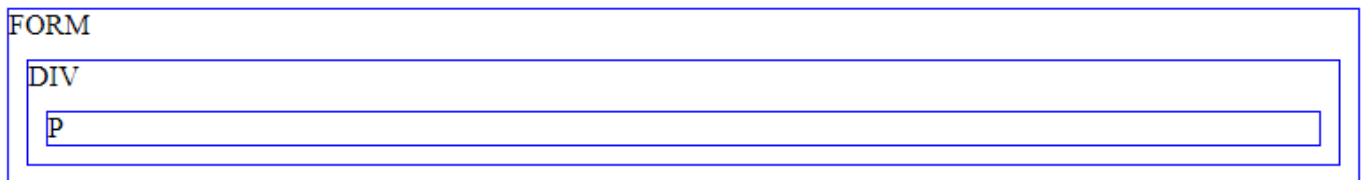
Il **primo** argomento rappresenta l'evento da gestire

L'**ultimo** argomento rappresenta la procedura da eseguire in risposta all'evento.

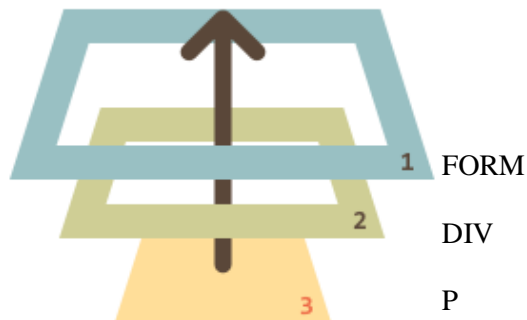
Event bubbling

Si supponga di avere la seguente condizione di elementi html annidati:

```
<form onclick="alert('form') ">FORM
  <div onclick="alert('div') ">DIV
    <p onclick="alert('p') ">P</p>
  </div>
</form>
```



Per default gli eventi in java script vengono propagati agli elementi del DOM tramite la cosiddetta **bubbling phase** ascendente, cioè dall'elemento più interno verso l'elemento più esterno:



Cioè:

- Se si clicca sulla form compare il messaggio **form**
- Se si clicca su tag DIV compaiono i messaggi **div** e **form**
- Se si clicca su tag P compaiono i messaggi **p** e **div** e **form**

Directly-bound events

Se viene omesso il 2° parametro del metodo `.on` (**selector**) (oppure impostato a `null`), l'event handler corrispondente viene definito **directly-bound-event**.

In questo caso il gestore di evento viene richiamato ogni volta che l'evento si verifica, **sia in modo diretto sull'elemento stesso, sia quando l'evento si verifica su un elemento sottostante nel DOM e viene propagato verso l'alto mediante la bubbling phase.**

Nel caso dei directly bond events definire l'associazione tramite il metodo `on` oppure in forma diretta è sostanzialmente la stessa cosa. Rimane il fatto che con `on` è possibile poi eseguire `off`.

Delegated events

Ritornando all'esempio precedente, **si supponga di gestire l'evento soltanto sulla form**. Quando l'evento si genera su un elemento interno, viene propagato fino alla form e, insieme all'evento, viene propagato anche il puntatore all'elemento che ha generato l'evento. Occorre però definire una apposita sintassi per potervi accedere. Impostando come 2° parametro del metodo `.on` il selettore ad un elemento interno alla form, se l'evento è stato generato su quell'elemento la form **NON gestisce l'evento** e lo rimanda giù (cioè lo **delega**) all'elemento interno (**capturing phase**) che provvederà alla relativa gestione. Si parla in questo caso di *delegated-event*, cioè la form delega la gestione dell'evento all'elemento interno che lo ha generato. Es

```
$("#form").on("click", "p", function() {  
    alert($(this).text());  
});
```

Se l'evento `click` si verifica su un tag `p`, la form non gestisce l'evento ma lo rimanda all'elemento medesimo. All'interno di questa funzione `this` rappresenta il puntatore all'elemento che ha effettivamente scatenato l'evento.

Vantaggi:

- Al momento della definizione dell'evento è sufficiente che esista il contenitore (al limite si possono usare document oppure body). **L'evento verrà poi intercettato da qualunque elemento interno eventualmente aggiunto dinamicamente al DOM in un secondo tempo.** A tale proposito l'evento `$(document).ready()` non ammette la forma `.on` in quanto, trattandosi dell'elemento più alto del DOM, non ha senso **catturare** eventi provenienti da elementi superiori
- Un altro vantaggio è quello di poter eseguire infiniti eventi con un'unica associazione di evento. Supponendo ad esempio di avere una tabella con 1000 righe, nel primo dei seguenti casi vengono definite 1000 associazioni di evento, mentre nel secondo caso viene definita una sola associazione di evento:

```
$( "#dataTable tbody tr" ).on( "click", function() {  
    console.log( $( this ).html() );  
})
```

Può essere riscritto nel modo seguente:

```
$( "table" ).on( "click", "tr", function() {  
    console.log( $( this ).html() );  
});
```

Svantaggio

- L'unico limite dei delegated events consiste nel fatto che non è possibile rimuovere l'evento soltanto su alcuni elementi del gruppo. Il metodo `off()` accetta anch'esso un secondo parametro **selector**, che però deve coincidere con il selettore usato all'interno del metodo `on()`. O si rimuovono **tutti** gli eventi o **nessuno**. Un rattoppo (*workaround*) potrebbe essere il seguente:

```
$(document).on('click', '.btn', function () {  
    alert('pressed');  
    $(this).removeClass("btn");  
});
```

Passaggio di parametri alla funzione di evento

Il **terzo parametro** del metodo `.on` consente di passare dei parametri json al gestore di evento.

Il gestore di evento riceve infatti sempre un parametro facoltativo **event** il cui campo `data` contiene gli eventuali parametri passati in formato json

```
$("#form").on("click", "p", {"msg":"ciao"}, function(event) {  
    alert(event.data.msg); });
```

Introduzione a jQueryUI

Le Librerie **jQueryUI** forniscono un insieme di metodi per l'interfaccia grafica della pagina. Questi metodi possono essere suddivisi sostanzialmente in tre gruppi:

- **effetti grafici** aggiuntivi rispetto a quelli di jQuery
- **widget** cioè controlli complessi come calendari, finestre modali e navigazione a schede.
- **interazioni** complesse come ordinamento di elementi, drag and drop, etc.

Ultime versioni (data rilascio ufficiale)

1.9.2	23 novembre 2012
1.11.4	13 dicembre 2015

Il sito **jqueryui.com** mostra l'elenco completo di tutti i vari componenti disponibili.

La sezione **API documentation** mostra tutte le singole proprietà e metodi.

La sezione **download** consente di personalizzare il download con la possibilità di scaricare soltanto i componenti che si intende utilizzare all'interno dell'applicazione, in modo da alleggerirla il più possibile. E' anche possibile scegliere un tema grafico da associare ai vari componenti. Se si esegue il download dalla pagina principale senza selezionare un tema, viene automaticamente scaricato il tema "**Base**".

Themes

Nella sezione **download**, a fine pagina, è possibile scegliere un tema grafico da abbinare alla pagina. Come default è impostato il tema "**Base**" con sfondi di colore grigio. Per vedere in anteprima i colori di un tema occorre selezionare il tema e cliccare sul link [design a custom theme](#) che apre un **Theme Roller** basato sul tema selezionato e, tramite il theme roller in alto a sinistra, consente di personalizzare tutti le singole impostazioni. Sono abbastanza luminosi (sfondi chiari) **UI lightness, flick, pepper, grinder**

Utilizzo

Come per le librerie jQuery si può utilizzare un collegamento onLine oppure usarle in locale.

```
<link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.min.css" />
<script type="application/javascript" src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js">
```

All'interno dell'applicazione occorre specificare un link ai files **jquery-ui.css** e **jquery-ui.js**.

jquery-ui.css definisce l'aspetto grafico dei vari componenti (sulla base del tema scelto) ed è l'unione dei due files **jquery-ui.structure.css** + **jquery-ui.theme.css**.

Il primo contiene la struttura degli oggetti (dimensione, posizione, margini, etc) mentre il secondo contiene gli elementi di colore dipendenti dal tema (font-family, colore del testo, colore dello sfondo, etc).

NB

- I valori delle property numeriche **DEVONO** essere scritti in modo diretto, **SENZA apici e SENZA unità di misura**.
- Le classi jQueryUI possono essere applicato **SOLO** ad elementi presenti sulla pagina (cioè dopo che sono stati appesi al DOM).

Effects

Elenco dei principali stili assegnabili ai vari elementi

ui-widget-header	intestazione dell'elemento Testo Bianco grassetto su IMG di sfondo
ui-widget-content	corpo dell'elemento. Testo #333 su IMG di sfondo grigia #eee
ui-corner-all	generico, spigoli arrotondati

stili applicabili ai pulsanti / collegamenti ipertestuali

ui-state-default	Normale pulsante cliccabile
ui-state-disabled	pulsante disabilitato (viene rimosso lo stato .active, per cui il pulsante non risponde più al click)
ui-state-hover	mouse over.
ui-state-active	elemento attualmente attivo.

Attenzione che spesso queste classi hanno una immagine di sfondo che rende impossibile per l'utente cambiare il colore di sfondo. A tal fine occorre necessariamente impostare **background-image:none**

Animazioni sui colori

La libreria UI estende le capacità del metodo `.animate()` nativo, introducendo le transizioni sul **colore** che permettono di sostituire un colore con un altro con un effetto graduale.

Le proprietà CSS che supportano questo effetto sono: `backgroundColor`, `borderBottomColor`, `borderLeftColor`, `borderRightColor`, `borderTopColor`, `color`, `outlineColor`.

Animazioni sulle classi

La libreria UI estende il metodo `addClass()` aggiungendo 3 nuovi parametri che consentono di applicare le proprietà definite nella nuova classe mediante un transizione temporale (in modo simile alla sintassi del metodo `animate()`).

```
$("#box").addClass("nuovaClasse", 1000, "easeOutBounce", callback );
```

Il 2° parametro **duration** indica la durata della transizione (espressa in msec).

Il 3° parametro **easing** permette di modificare la dinamica con cui si svolge l'animazione, simulando effetti elastici o di attrito al fine di rendere più realistica l'animazione. I valori possibili sono quelli derivati dai CSS3: `ease`, `easeOut`, `easeIn`, `easeInOut`, `linear` con moltissime combinazioni.

Elenco completo ed esempio all'indirizzo <http://gsgd.co.uk/sandbox/jquery/easing/>

Il 4° parametro **complete** permette di richiamare una funzione che verrà eseguita al termine dell'animazione.

Gli stessi parametri possono essere applicati anche ai metodi `switchClass()` e `removeClass()`

```
$("#box").switchClass("class1", "class2", 1000, "easeOutBounce", callback);
```

Il metodo effect()

Consente di eseguire moltissimi effetti passati semplicemente come stringa.

Il secondo parametro indica eventuali opzioni aggiuntive per il controllo dell'animazione (es 60 frammenti)

Il terzo parametro indicare la durata in millisecondi dell'effetto,

Il quarto parametro indica una eventuale funzione di *callback* da eseguire **al termine** dell'animazione. Es

```
ref.effect( "explode", {pieces: 60}, 900, function(){.....}); // oppure
ref.hide( "explode", {pieces: 60}, 900, function(){.....});
```

Le classi di interazione

Le cosiddette classi di interazione, come ad esempio `resizable()` `draggable()` `droppable()` `selectable()` sono applicabili a qualsiasi elemento HTML e consentono di associare a tale elemento particolari caratteristiche, come ad esempio quella di diventare ridimensionabile, oppure trascinabile, oppure selezionabile.

Il richiamo di queste classi è da intendersi come il richiamo di un **costruttore** che istanzia la classe e la associa al tag HTML indicato dal selettore.

Tutti i costruttori delle classi di interazione accettano come parametro un elenco di **opzioni**

- racchiuse all'interno di parentesi graffe
- separate da virgola
- espresse nel formato **nome:valore** (senza virgolette per i valori numerici)

Queste opzioni possono essere di due tipi:

- **Proprietà** `maxHeight:250`
- **Eventi** `start:inizializza`
- Gli elementi sui quali si sta svolgendo un certo evento, implementano automaticamente delle specifiche **Classi UI** che possono essere utilizzate nel programma per appositi scopi.

Gli Eventi

Quasi tutte le classi di interazione sono dotate degli eventi:

start richiamato nel momento in cui ha inizio l'interazione (ad esempio il trascinamento)
stop richiamato nel momento in cui l'interazione ha termine
create richiamato nel momento in cui la classe di interazione viene istanziata tramite il costruttore
più eventuali altri eventi legati alla specifica interazione (ad esempio **DRAG** per la `draggable`, **DROP** per la classe `droppable`, **RESIZE** per la classe `resizable`).

Le Proprietà

Le proprietà definite all'interno del costruttore (ad esempio `cursor:"move"`), vengono :

- automaticamente applicate in corrispondenza dell'evento **start**
- automaticamente rimosse in corrispondenza dell'evento **stop**.

Una volta istanziato l'oggetto, le Property possono essere lette / modificate tramite la voce **Option** :

```
$("#box").resizable ({maxWidth: 350});  
$("#box").resizable("option", "maxWidth", 360);  
$("#box").resizable("option", "disabled", true);
```

In alternativa si può richiamare una seconda volta il costruttore.

Le proprietà precedenti vengono mantenute e ad esse vengono aggiunte quelle nuove.

Nel caso di proprietà a valore multiplo (es `border`), all'interno di **Option** si scrive esattamente ciò che si scriverebbe all'interno della Property impostata direttamente.

Tutte le classi dispongono della **Property disabled** ma non della Property `enabled`.

Gestione degli Eventi

Alle funzioni di evento occorre assegnare un **Event handler** cioè un **Riferimento** alla funzione di evento da eseguire in corrispondenza dell'evento. Esempio:

```
$("#box").resizable({
    maxHeight: 250,
    maxWidth: 350,
    start: inizializza,
    resize: function() {.....},
    stop: termina
})
```

Al solito le funzioni di evento possono essere scritte in loco in forma anonima oppure possono essere scritte esternamente mediante una apposita funzione.

```
var inizializza = function(event, args) { . . . . . };
```

Parametri

Le funzioni di evento hanno sempre **2 parametri: event e args** che ricevono automaticamente dal sistema nel momento in cui scaturisce l'evento. Questi parametri possono anche essere omessi se non sono utilizzati nel successivo codice di gestione dell'evento.

Il parametro event è simile al sender di C#. Rappresenta un puntatore ad un oggetto event che fornisce alcune informazioni sul tipo di oggetto che ha scatenato l'evento. Fra le proprietà ci sono :

.target Puntatore javascript all'elemento del DOM che ha scatenato l'evento (equivalente a **this**). L'elenco complete di tutti i valori è riportato a pagina 9

Il parametro args rappresenta un elenco di argomenti che, come in C#, dipendono dallo specifico evento.

Se all'interno delle funzioni di evento non si usano i parametri **event** e **args**, allora tali funzioni possono anche essere richiamate esplicitamente come una normalissima funzione priva di parametri.

Definizione di un gestore di evento fuori del 'costruttore'

Il gestore di evento, oltre che all'interno del costruttore (insieme alle Property), può essere definito anche esternamente attraverso il metodo **.on()** che è equivalente del metodo **.on** di jQuery base.

Bisogna però fare attenzione che il nome da passare a **.on()** è dato dal nome della classe seguito dal nome dell'evento (tutto minuscolo). Nel caso dell'evento **start** della classe **resize** occorrerà scrivere:

```
$("#myDiv" ).on( "resizestart", function(event, args) { } );
$("#myDiv" ).on( "dragstop", function(event, args) { } );
```

Gestione dei Metodi

Tutte le varie classi di jQueryUI, oltre a proprietà ed eventi, dispongono anche di diversi metodi riutilizzando il nome del costruttore e passandogli come parametro il nome del metodo racchiuso tra apici. Questi metodi sono chiaramente utilizzabili solo **DOPO** che la classe è stata istanziata, mentre in fase di istanza si può passare soltanto un elenco di **Proprietà** ed **Eventi**. Esempio di Metodi:

```
$("#box").resizable ("disable"); Disabilita temporaneamente la funzionalità indicata
$("#box").resizable ("enable"); Riabilita una funzionalità temporaneamente disabilitata
```

```
$("#box").resizable ("destroy"); Dealloca l'oggetto resizable
$('#box').dialog('open'); Apre la finestra di dialogo
```

Occorre distinguere il Metodo **disable** dalla Property **disabled** disponibile in tutte le classi. Le due cose, pur con sintassi diverse, sono equivalenti. Attenzione però che la Property **enabled** non esiste.

NOTA: Le presenti classi di interazione devono essere applicate agli oggetti della pagina HTML soltanto **DOPO** che l'elemento (eventualmente creato dinamicamente) è stato appeso al DOM.

La classe RESIZABLE ()

```
$("#box").resizable({opzioni});
```

Come per tutti i metodi successivi, il semplice richiamo del metodo **.resizable()**, (anche senza opzioni, **nel qual caso le parentesi graffe possono essere omesse**), rende ridimensionabile l'elemento a cui viene applicato.

Proprietà:

```
aspectRatio: true, /* mantiene le proporzioni */
maxHeight: 350,
maxWidth: 350,
minHeight: 150,
minWidth: 150,

animate: true,
// il ridimensionamento viene eseguito tramite un'animazione,
// che però parte solo nel momento in cui si rilascia il mouse
ghost: true,
// ha senso solo abbinato alla precedente
// crea un helper semitrasparente che fa da traccia durante il resize
```

Eventi:

START = inizio ridimensionamento
RESIZE = ridimensionamento in corso
STOP = ridimensionamento terminato

La classe DRAGGABLE()

Il metodo **.draggable()** associato ad un elemento, lo rende trascinabile alla pressione del mouse.

```
$("#box ").draggable({opzioni});
```

axis: "x"	Consente il trascinamento solo lungo l'asse x
axis: "y"	Consente il trascinamento solo lungo l'asse y
grid: [20,20]	Si muove "a step" di 20 pixel per volta.
cursor: "move"	Imposta la forma del cursore del mouse (css property cursor)
cursorAt: {top:56, left:56}	Posizione del cursore durante il trascinamento
containment: "#wrapper"	Consente il trascinamento solo all'interno del box indicato
scroll: true (default)	Se il genitore è overflow auto, quando si trascina un elemento contro i bordi del genitore, il genitore esegue uno scroll.

stack: "#wrapper div"

Fa sì che l'elemento trascinato assuma uno **z-index** maggiore rispetto a tutti gli elementi indicati nel parametro, e dunque venga visualizzato al di sopra di ogni altro elemento. **L'impostazione di questa proprietà inoltre modifica tutti gli z-index degli elementi indicati nel parametro.** Nello specifico sembra che lo **aumenti** di una o due unità (lasciandolo valorizzato anche al termine del trascinamento). Se un elemento deve stare dietro ad altri occorre impostare z-index=-1/-2. Accetta come parametri sia puntatori JS che JQ

revert: **true**

Al termine del trascinamento, l'elemento ritorna **sempre** nella sua posizione iniziale. (Se però viene fatto il Drop su un'area Droppable, l'evento Drop viene comunque generato)

: **false**

L'elemento può essere rilasciato ovunque. Non ritorna **mai** indietro.

: **"invalid"**

L'elemento draggable viene rispedito indietro (nella sua posizione originale) solo se viene rilasciato in un'area **non** Droppable

: **"valid"**

Al contrario di prima, l'elemento draggable viene rispedito indietro nel momento in cui viene rilasciato in un'area Droppable.

snap: true

Si muove soltanto in modo "calamitato" rispetto agli altri elementi trascinabili. E' possibile passare come parametro un **selettore CSS** nel qual caso l'elemento sarà calamitato rispetto a quell'elemento

helper: "clone" **original** / **clone** Impostando **clone**, non viene trascinato l'oggetto Draggable base, ma viene creata una copia (**helper**) e viene trascinata la copia. Al momento del rilascio, in caso di "revert":true il clone ritorna nella posizione iniziale. In caso di invece di "revert":false **al momento del rilascio il clone viene rimosso dal DOM.** Se si desidera avere una copia del clone nella posizione di rilascio occorre creare una nuova istanza del clone(**clone del clone** Vedere evento **DROP** della classe **DROPPABLE**) Se si appende il clone stesso al DOM, ciò che viene appeso è in realtà l'elemento originale, che viene **tagliato** dalla sua posizione originale ed incollato nella nuova posizione. Il clone del clone **NON** è draggable, per cui non è necessario disabilitare il draggable

Nota: Il clone è un oggetto **temporaneo** che eredita le classi applicate all'elemento base, ma **NON** eredita le proprietà CSS applicate staticamente al solo ID dell'elemento base. Per cui se si desidera che il clone assuma tutte le caratteristiche dell'elemento base, queste caratteristiche dovranno essere definite non sull'ID ma mediante una classe. Le CSS property impostate dinamicamente sull'ID vengono invece normalmente ereditate.

- Se le proprietà vengono assegnate sull'ID, l'elemento trascinato sarà a sfondo trasparente, per cui sembra che venga trascinato soltanto il testo.
- Se si desidera che il clone trascinato abbia un aspetto diverso rispetto all'oggetto base, è possibile assegnare a helper una funzione **custom**. Es:

```
helper: function() {
    var html=$( "<div></div>" );
    html.css({border:"1px solid black",
        width:"120px", height:"25px",
        backgroundColor:"#ffa",
        textAlign:"center" });
    html.text( $(this).text() );
    return html;
}
```

- **In caso di clone creato manualmente mediante una funzione personalizzata**, si può appendere direttamente il clone all'interno del contenitore desiderato senza ulteriore clonazione.

Eventi:

START = inizio trascinamento. Il parametro **args** presenta le seguenti proprietà:

args.originalPosition.left

args.originalPosition.top coordinate iniziali al momento dell'inizio del trascinamento

DRAG = trascinamento in corso

STOP = trascinamento terminato con rilascio del pulsante del mouse. In caso di **revert**, l'evento stop viene generato **NON** al rilascio del mouse, ma **al TERMINE del revert** (quando l'oggetto è completamente rientrato nella sua posizione originale)

L'evento **DRAG** presenta, all'interno del parametro **args**, due interessanti proprietà:

- **args.position** – posizione attuale dell'elemento trascinato { left, top } relativamente alla pagina oppure relativamente all'elemento contenitore se questo è **position:relative**. Può essere usato anche in scrittura per modificare la posizione.
- **args.helper** – riferimento all'oggetto helper in corso di trascinamento. Questo riferimento a sua volta dispone di una proprietà **position** che contiene le coordinate { left, top }, relative alla posiz corrente

Classi:

Gli elementi in corso di trascinamento implementano automaticamente la classe

.ui-draggable-dragging

I Metodi enable e disable

Il trascinamento può essere disabilitato / riabilitato mediante i metodi **enable** o **disable** che possono essere richiamati soltanto **dopo** che il **.draggable()** è stato istanziato.

Il **draggable("disable")** dopo aver disabilitato l'oggetto, gli assegna un livello di trasparenza che può risultare fastidioso. Per evitare questo effetto, dopo aver disabilitato il trascinamento, applicare la seguente Proprietà di stile che assegna all'oggetto un aspetto SOLIDO pieno:

```
ref.css("opacity",1);
```

La classe DROPPABLE()

Il metodo **.droppable()** associato ad un oggetto fa sì che quell'oggetto possa "ricevere" un elemento draggable. In corrispondenza del rilascio del draggable viene generato l'evento **DROP**

```
$("selettore").droppable({opzioni});
```

Proprietà:

- activeClass:"className"** Classe che da applicare all'elemento Droppable quando è in corso il Drag di un elemento valido per il Drop sull'oggetto corrente
- hoverClass:"className"** Classe che deve essere applicata all'elemento Droppable nel momento in cui un elemento draggable "entra" nell'area dell'elemento Droppable
- accept: "Selector"** Accetta in DROP SOLO gli elementi identificati dal corrispondente selettore CSS. Per default accetta in DROP qualsiasi elemento. Attenzione che impostando **accept**, **activeClass** si attiverà solo per gli elementi validi impostati all'interno di **accept**.
E' anche possibile assegnare una funzione che ritorni TRUE se l'elemento è accettato oppure FALSE in caso contrario:
accept: function() {
 if() return true;
 else return false; **},**

Eventi:

DROP = rilascio dell'oggetto, Il parametro **args** contiene le seguenti proprietà:

- **args.draggable** rappresenta un puntatore **jQuery** all'elemento draggable **che ha scatenato l'evento DROP**. (attenzione: puntatore jQuery a differenza di **event.target** che è un javascript)
- **args.position** – posizione attuale dell'elemento trascinato sull'elemento droppable (relativamente alla pagina oppure relativamente all'elemento contenitore se questo è **position:relative**). Può essere usato anche in scrittura per modificare la posizione.
- **args.helper** In caso di clone rappresenta un puntatore jQuery al clone in fase di trascinamento. In caso di clone **args.draggable** punta all'elemento da cui ha avuto inizio il drag (dunque dispone dell'eventuale ID), mentre **args.helper** punta al clone per cui NON dispone dell'ID. Per clonare il clone : **var _clone = args.helper.clone() ;**

A parte questo, tra **args.draggable** e **args.helper** ci sono altre piccole differenze :

- Sia l'elemento puntato da **draggable** che l'elemento puntato da **helper** implementano le classi / caratteristiche css implementate dall'elemento da cui ha avuto origine il clone
- L'elemento puntato da **draggable** mantiene queste property anche DOPO il DROP, anche nel caso di elemento personalizzato creato tramite funzione custom. L'elemento puntato da **helper** in caso di funzione custom, dopo il drop l'oggetto non assume le caratteristiche dell'oggetto originale ma mantiene quelle dell'oggetto custom
- Se, dopo l'inizio del trascinamento, si applicano nuove property all'elemento originale (es visibility:"hidden") queste property si ripercuotono sull'elemento puntato da **draggable**, mentre non hanno alcun effetto sull'elemento puntato da **helper**
- Dopo il Drop l'elemento puntato da **draggable** assume il valore **position:static** (normale flusso). Dopo il Drop l'elemento puntato da **helper** assume automaticamente il valore **position:absolute** (rispetto alla pagina) e due valori di top e left pari alla posizione corrente. Se il contenitore è position:relative top e left vengono interpretati come assoluti rispetto al contenitore, per cui traslano l'elemento lontanissimo dalla posizione di drop. In entrambi i casi questi posizionamenti sono abbastanza fastidiosi perché rendono problematiche eventuali animazioni, a meno che il bersaglio sia anche lui position absolute

La classe SELECTABLE ()

Il metodo **.selectable()** **deve** essere applicato ad un contenitore e rende selezionabili singolarmente o in gruppi tutte gli elementi **interni** al contenitore.

Eventi:

START = eseguito nel momento in cui inizia la selezione/deselezione di un elemento (click del mouse)

SELECTING = eseguito nel momento in cui un elemento è in fase di selezione

SELECTED = eseguito nel momento in cui un elemento è stato selezionato

STOP = eseguito nel momento in cui viene terminata la selezione degli elementi (rilascio del mouse).

E' l'evento migliore da gestire a livello di codice. La differenza rispetto a **selected** è che, se seleziono n elementi tramite un rettangolo, **selected** viene generato n volte, **stop** una volta sola

Classi:

Agli elementi dichiarati **selectable** vengono **automaticamente** applicate 2 classi il cui contenuto (colori, immagini, etc.) non è definito all'interno degli UI-CSS ma **deve** essere definito dall'utente nel proprio CSS

- **ui-selecting** applicata all'elemento in fase di selezione
- **ui-selected** applicata a tutti gli elementi attualmente selezionati

Queste due classi possono essere :

- modificate a livello di CSS con l'aggiunta di property personalizzate (es background-color)
- utilizzate all'interno dell'evento stop per accedere a tutti gli elementi selezionati.
- aggiunte manualmente da codice per eseguire delle selezioni direttamente da codice.


```
$("#voci li").each(function (i, ref ) {
    if((i%2)==0)    // seleziono voci pari
        $(ref).addClass("ui-selected");
});
```

Nota sulla Selezione e Trascinamento di più oggetti

Se si definisce un contenitore come **selezionabile** (che significa che tutti gli elementi interni saranno selezionabili), e poi si definiscono tutti i vari elementi anche **trascinabili**, le due azioni possono cooperare, però :

- Il click è intercettato come inizio trascinamento, per cui la selezione multipla è possibile solo attraverso il rettangolo.
- Inoltre in corrispondenza del click si perderebbe comunque anche la selezione multipla e verrebbe trascinato soltanto l'elemento corrente.
- Per trascinare in gruppo tutti gli elementi selezionati occorre :
 - disabilitare inizialmente il **DRAGGABLE** ed abilitare soltanto il **SELECTABLE**.
 - In corrispondenza dell'evento **STOP** del **SELECTABLE** applicare a tutti gli elementi **.ui-selected** una nuova classe, denominata ad esempio "gruppo", ed abilitare il **DRAGGABLE** sull'intero gruppo (infatti la classe **.ui-selected** viene rimossa in corrispondenza del click relativo al **DRAG**)
 - In corrispondenza dell'evento **DRAG** occorre leggere costantemente l'offset dell'oggetto corrente e replicarlo manualmente su tutti gli elementi del "gruppo" (escluso l'elemento corrente).

La classe SORTABLE ()

Il metodo `.sortable()` **deve** essere applicato ad un contenitore.
Consente di Spostare tutti gli elementi **interni** al contenitore.

Classi:

Agli elementi dichiarati sortable viene automaticamente associata la classe:

- **ui-sortable-helper** applicata all'elemento in fase di trascinamento

Proprietà:

items:"selector"	Consente di abilitare al trascinamento soltanto quegli elementi che implementano la classe indicata dal selettore.
cancel:"selector"	Consente di inibire al trascinamento gli elementi che implementano la classe indicata dal selettore. Applicando ad un elemento la classe ui-state-disabled , questo oltre a diventare non trascinabile, assume l'aspetto grafico della classe ui-state-default . Con cancel invece si può impostare un aspetto grafico personalizzato tramite una classe utente.
placeholder:"className"	Consente di visualizzare un segnaposto nella posizione in cui sta per essere trascinato l'oggetto corrente. A questo segnaposto si può applicare una user Class oppure la classe ui-state-highlight . Il parametro è di tipo className , per cui va scritta SENZA PUNTINO.

Metodi:

- .cancel** = annulla l'ultimo cambiamento avvenuto nella lista (tipico undo).
- .toArray** = restituisce un **vettore** con gli id delle voci interne al wrapper, nell'ordine in cui si trovano

Eventi: **SORT** = richiamato durante il trascinamento di ciascun oggetto.

Widget

I Widget utilizzano la stessa sintassi dei metodi di interazione. Vengono istanziati tramite un costruttore al quale è possibile passare Proprietà / Eventi.

Tutti i widget implementano la classe **.ui-widget**

La classe button()

Il widget `button` è associabile ad un qualsiasi tag HTML per trasformarlo in un pulsante

```
$("#box").button();
```

La riga precedente è completamente equivalente alla seguente:

```
$("#btn").addClass("ui-button ui-widget ui-corner-all");
```

Proprietà:

disabled: `true/false`

label: testo da visualizzare nel pulsante

Se l'opzione `label` non viene impostata, nel caso del `button` viene assunto come testo il value del button; nel caso di `Radio` e `CheckBox` viene assunto come testo il contenuto della label HTML associata al pulsante

icon: `"ui-icon-newwin"`

L'opzione **icon** consente di visualizzare una icona a sinistra oppure a destra del testo del pulsante. Si aspetta come parametro un **className** scritto come stringa (senza puntino). La classe **ui-icon** imposta come immagine di sfondo `ui-icons_444444_256x240.png` che contiene nello stesso file moltissime icone 16x16. La classe **ui-icon-name**, mediante la proprietà `backgroundPosition`, individua una singola icona all'interno del file. Se la classe indicata non esiste viene visualizzata la prima icona dell'elenco (quella con indice 0,0).

iconPosition: `"end"`

Consente di visualizzare l'icona a destra del testo (default = sinistra)

showLabel: `true/false` // Visualizza / Nasconde il testo del pulsante

E' anche possibile associare icone personalizzate definendo una classe personale del tipo:

```
.ui-icon.myIcon { background-image: url(myIcon.gif) }
```

e poi impostare la property **icon** : `"myIcon"`

Classi:

- I pulsanti a riposo implementano la classe **ui-state-default** (con immagine di sfondo)
- I pulsanti premuti implementano la classe **ui-state-active** (con immagine di sfondo).
- In corrispondenza del mouse over viene applicata la classe **ui-state-hover**

Le classi `controlgroup()` e `checkboxradio()`

Controlgroup, applicato ad un **contenitore**, applica una trasformazione grafica a tutti i controlli contenuti in quel contenitore. Particolarmente adatto per i **RadioButton**.

Nella versione **1.12** è stato aggiunto un nuovo widget **Checkboxradio** identico al precedente ma applicabile NON al contenitore ma ai controlli interni. Entrambi questi widget vanno bene per i **Radiobutton** ma non per i **Checkbox** per i quali è graficamente preferibile il widget **Button**).

La classe autocomplete()

Consente di fornire all'utente, durante l'inserimento dati in un Text Box, una serie di suggerimenti memorizzati all'interno di un normalissimo vettore java script. In base al tasto premuto, vengono mostrati solo i suggerimenti che iniziano con quella lettera o che contengono quella lettera.

```
var suggerimenti = ["C","C++","COBOL",...];
$("#txtLinguaggi").autocomplete({ source: suggerimenti });
```

La classe dialog()

Può essere assegnato ad un qualsiasi elemento della pagina HTML (tipicamente un tag DIV).

```
$("#myDIV").dialog();
```

A seguito dell'assegnazione il tag "myDIV" si *trasforma* in un Dialog Box.

Come titolo della finestra viene utilizzato il contenuto dell'attributo **title** del tag myDIV.

```
$('#myDIV').dialog({
    width:"400", height:"300",           // omettere px !!
    autoOpen:false,
    show:"blind",
    hide:"explode"
});
```

Proprietà:

- autoOpen:** false Il dialog box, per default, viene aperto immediatamente in fase di istanza. Per evitare l'apertura occorre impostare la proprietà **autoOpen** : false
- width / height:** Dimensioni della finestra. Se si omettono vengono impostate automaticamente
- title:** Indica il testo da usare come titolo della finestra. In alternativa viene usato l'attributo HTML **TITLE** dell'elemento a cui è applicato il metodo dialog().
- draggable / resizable:** false Impostando su false queste opzioni, la finestra diventa rispettivamente fissa oppure non ridimensionabile. Può essere utile per rendere meno pesante in termini di risorse la visualizzazione di piccoli messaggi, che non necessitano di essere ridimensionati o spostati;
- modal:** (booleano - false) rende la finestra un controllo modale. In questo modo tutti gli elementi esterni saranno bloccati fino alla chiusura della finestra. L'uso dell'opzione crea un elemento semitrasparente di overlay grande come il documento HTML fra l'interfaccia e la finestra;
- show / hide :** Consentono di associare un effetto all'apertura e chiusura della finestra. Si può specificare un **effetto**, ed eventualmente una **durata**. Esempio:
show: "blind",
hide: { effect: "explode", duration: 1000 }
- buttons :** Consente di inserire nella finestra dei pulsanti aggiuntivi, ciascuno con un proprio **nome** e con una propria **funzione** di callback. Esempio:
buttons: {
 "Ok" : **function**() {
 eseguioperazione("premuto tasto OK");
 \$(this).dialog('close');
 },
 "Annulla" : **function**() {
 eseguioperazione("premuto tasto Annulla");
 \$(this).dialog('close');
 }
 }

Eventi:

open / close: Lanciati quando la finestra si apre o chiude.

Metodi open / close :

Questi metodi consentono di aprire / chiudere la finestra (già creata in precedenza), in corrispondenza di particolari eventi.

```
$("#myDIV").dialog("open"); // Consente di aprire la finestra DOPO che è stata creata
$("#myDIV").dialog("close"); // Consente di chiudere la finestra
$("#myDIV").dialog("moveToTop"); // La finestra viene portata in primo piano
$("#myDIV").dialog("isOpen"); // Restituisce true se la finestra è aperta
```

La classe accordion()

Il widget Accordion si applica su una struttura HTML del tipo:

```
<div id="wrapper">
  <h2> Prima Sezione</h2>
  <div> contenuto </div>
  <h2> Seconda Sezione </h2>
  <div> contenuto </div>
</div>

$("#wrapper").accordion({
  active:2,
  collapsible: true,
  event:"click" });
```

Proprietà:

collapsible:true (default false) Fa sì che un click su una scheda aperta la richiuda, passando in uno stato con tutte le finestre chiuse.

active:2 n° della scheda da visualizzare (**a base 0**).
Impostando **active:false** oppure **active:"none"** tutte le schede verranno chiuse. Se però si parte da una situazione di schede tutte chiuse, quando le schede verranno aperte avranno una altezza valutata automaticamente con comparsa della scroll bar laterale. Per rimediare occorre impostare una altezza esplicita ai DIV delle varie schede.

event:"click" Indica l'evento in corrispondenza del quale il widget apre e chiude le sezioni. Il default è **click** ma si possono usare altri eventi (**mouseover** o **mouseout**)

Eventi:

beforeActivate: Richiamato nel momento in cui ha inizio un cambiamento di scheda.
activate: Richiamato al termine dell'animazione relativa al cambiamento di scheda.

Per entrambi questi eventi args contiene i seguenti argomenti:

- .newHeader:** un riferimento all'header attivo;
- .oldHeader** un riferimento all'header attivo in precedenza;
- .newPanel:** un riferimento al contenuto attivo;
- .oldPanel:** un riferimento al contenuto attivo in precedenza

Se non c'erano schede aperte **oldHeader** e **oldPanel** sono nulli

Un elemento definito **Accordion()** può anche essere definito **Sortable()** nel qual caso le varie sezioni diventano trascinabili e riordinabili.

La classe tabs ()

Un tag **#wrapper** principale funge da contenitore per l'intero controllo. Un tag **UL** interno funge da barra dei menu, in cui ogni voce **LI** contiene un link ad un tag **DIV** corrispondente al contenuto di quella specifica scheda

```
$("#wrapper").tabs({
    active:2
});
```

Proprietà

- active: 1** indice della scheda da visualizzare (a base 0).
- collapsible: true** (default false) Fa sì che un click sul Titolo di una scheda aperta la richiuda, passando in uno stato con tutte le schede chiuse.
- event: "click"** Indica l'evento in corrispondenza del quale il widget apre e chiude le schede. Il default è **click** ma si possono usare altri eventi come `mouseover` o `mouseout`

Metodi

- refresh:** aggiornamento dell'oggetto Tabs (dopo che sono state eventualmente applicate variazioni da codice).

La classe datepicker ()

Apri un calendario che consente di scegliere una data.

```
$("#datepicker").datepicker({
    showOn: "button",
    buttonImage: "calendar.gif",
    buttonImageOnly: true,
    numberOfMonths : [1,2],
    changeMonth : true,
    changeYear : true,
    showButtonPanel: true
});
```

Proprietà

- showOn:** Indica quando il datePicker si deve aprire. I valori possibili sono **"focus"**, **"button"**, **"both"**.
"focus" è il default ed il calendario viene mostrato nel momento in cui il text box riceve il fuoco
"button" mostra invece una piccola icona ed il calendario viene aperto sul click sull'icona
"both" mostra l'icona e apre il calendario sia sul click sull'icona sia quando il text box riceve il focus
- buttonImage:** Se showOn è impostato su button/both, indica il path dell'icona da visualizzare
- buttonImageOnly:** racchiude l'icona dentro un pulsante, rendendola un po' più piccola.
- changeMonth: true/false** All'interno del datePicker, in alto, viene mostrato un **comboBox** in cui l'utente può scegliere direttamente il mese
- changeYear: true/false** All'interno del datePicker, in alto, viene mostrato un **comboBox** in cui l'utente può scegliere direttamente l'anno.
- numberOfMonths: [1,3]**, Indica quante **righe** e quante **colonne** mostrare all'interno del datePicker. Impostando 1,1 viene mostrato un solo mese. Impostando 1,3 vengono mostrati 3 mesi su 3 colonne successive all'interno di una stessa riga
- showButtonPanel:** Visualizza un pannello inferiore in cui sono presenti i due pulsanti **Oggi** e **Chiudi**. **Oggi** riporta il cursore del datePicker sulla data corrente. Il testo di questi due pulsanti può essere personalizzato mediante le proprietà **currentText** e **closeText**

Classi:

- L'intestazione del datepicker implementa la classe **.ui-datepicker-header** la quale non ha immagini di sfondo e può essere utilizzata da CSS per modificare il colore di sfondo.

Impostazione della lingua

Per cambiare la lingua del datepicker, è possibile scaricare da jQueryUI.com le varie librerie linguistiche disponibili, librerie che poi devono essere collegate al file html. La libreria aggiunge la lingua indicata all'interno del vettore associativo `$.datepicker.regional[]`. Attenzione queste librerie, al termine, settano come lingua di default quella appena caricata.

```
$.datepicker.setDefaults($.datepicker.regional['it']);
```

Se si toglie questa riga, la lingua di default è l'inglese, la cui libreria è precaricata in jQueryUI.

In qualunque momento è possibile cambiare la lingua in uso mediante il seguente comando:

```
$("#datepicker").datepicker( "option", $.datepicker.regional["it"]);
```

Se si vuole cambiare la lingua in corrispondenza del `document.ready`, occorre linkare il file jQuery dopo rispetto al file della lingua (che altrimenti prevale).

La classe menu ()

Trasforma un tag UL contenente le varie voci di menù in un vero e proprio menù.

```
$("#menu").menu({
  select: visualizza,
  position: { my: "left top", at: "right-10 top+10" }
});
```

Proprietà

position: permette di impostare la posizione in cui verranno visualizzate le voci di secondo livello e successivi. Ad esempio:

```
position: {my:"left top", at: "right top" }
```

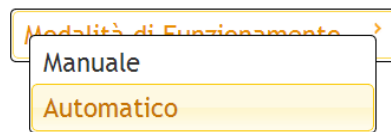
che significa che i valori **Left** e **Top** del menu secondario devono essere allineati ai valori **right** e **top** del menù di livello più alto (che peraltro rappresentano i valori di default)



Nota: Se vengono specificati solo i valori **MY** e non i valori **AT**, viene assunto come default di **AT** il valore **center**. Ad esempio:

```
position: { my: "left top", at: "left+10px" }
```

che significa che il top del menu secondario verrà allineato a metà del menu primario, producendo il seguente effetto:

**Eventi:**

select: Richiamato in corrispondenza della selezione di una qualsiasi voce di menù. Il parametro `args.item` contiene un puntatore alla voce selezionata (che è sempre un elemento LI). C'è però il problema che, nel caso di voci annidate, l'elemento **LI** selezionato contiene al suo interno anche il testo di tutte le voci annidate. La soluzione più semplice è quella di aggiungere ad ogni elemento **LI** un attributo fittizio denominato ad esempio **voce**, e andarlo poi a leggere dentro l'evento select.

```
alert(args.item.attr("voce")) ;
```

Classi:

Ai widget **menu** viene automaticamente associata la classe **.ui-menu** che viene applicata a tutti i menù di qualsiasi livello. Attenzione che i sottomenù ereditano la classe **.ui-state-active** applicata alla voce principale attiva, la quale imposta un colore del testo bianco assolutamente illeggibile. Si può però tranquillamente sovrascrivere questa classe applicando al testo il colore nero.

Metodi:

```
$("#menu").menu("collapse"); // Chiude il sottomenù attualmente aperto
$("#menu").menu("refresh"); // Rivisualizza il menu
```

Aggiunta di voci ad un menù

Mediante il solito metodo `.append()` è possibile aggiungere voci ad un menù di qualsiasi livello identificato mediante apposito ID.

```
$("#menu").append("<li><a href='#'>Nuova Voce</a></li>");
```

Se le voci interne non sono caratterizzate da un ID, è possibile accedere ad esse tramite i soliti metodi di navigazione di jQuery / CSS. Ad esempio:

```
var ref = $("#menu").find("ul:nth-of-type(1)");
```

La classe progressbar ()

Trasforma un generico tag DIV in una Progress Bar con valori compresi tra 0 e MAX. Non esiste MIN.

Proprietà

max Valore massimo (default 100)
value Utilizzato per impostare / leggere il valore corrente della Progress Bar.
 Disponibile anche sotto forma di metodo (ma solo in fase di scrittura)

```
$("#progressbar").progressbar('option', 'value', valore);
```

Assegnare il valore **false** alla property **value** equivale ad azzerare la Barra, con l'aggiunta però di un effetto grafico particolare (barra a strisce oblique).

Eventi

change Richiamato ogni volta che la barra cambia di valore. L'evento change vede il valore di **value** già aggiornato. **args non contiene nessun parametro.**
complete Richiamato quando il value diventa == max.

Colore interno

Quando si trasforma un tag DIV in una progress Bar, jQuery inserisce all'interno del tag DIV un altro tag DIV di cui fa variare la larghezza. Per modificare il colore di primo piano della Barra è sufficiente impostare il colore del tag div interno:

```
#progressBar>div { background:#0E0; }
```

Nel momento in cui viene impostato il valore false sulla proprietà value, la barra crea un ulteriore tag div interno di terzo livello nel quale carica una immagine con il tipico effetto strisciato. Per questo motivo il colore precedente va applicato soltanto sul figlio diretto #progressBar>div e non su figli e nipoti.

In realtà anche senza il segno di > funziona lo stesso in quanto l'effetto è applicato tramite immagine.

La classe slider ()

Trasforma un generico tag DIV in una Barra a scorrimento.

Proprietà

min/max	valori minimo e massimo impostabili con lo slider. I valori possono essere anche negativi. Default 0 / 100
range:	true / false . oppure stringa 'min' 'max' - se impostato su true , permette di gestire due cursori in modo da poter definire un intervallo di valori. - se impostato su min , lo slider sarà 'colorabile' nel tratto tra MIN – Value - se impostato su max , lo slider sarà 'colorabile' nel tratto tra Value - MAX
orientation:	stringa – 'horizontal'. Imposta l'orientamento dello slider, se impostato su 'vertical' la barra principale verrà mostrata verticalmente e i valori andranno dal minimo in basso al massimo in alto.
value:	intero – 0 Imposta il valore iniziale del cursore dello slider. Se usato insieme all'opzione range impostata su true, il valore passato sarà impostato sul primo cursore.
values:	array Accetta un array di due numeri con i quali impostare i valori iniziali dei cursori. values:[10,190], // a base 0 Impostando values compare automaticamente un doppio cursore anche senza impostare la proprietà range , che serve soltanto a migliorare l'aspetto grafico (colori) <pre>var values = \$("#selector").slider("option", "values"); \$("#selector").slider("option", "values", [10, 25]);</pre>
step:	intero. Imposta un intervallo di valori selezionabili. Default = 1

Eventi

slide:	lanciato durante il trascinamento del cursore. All'interno dell'evento slide il value non è perfettamente aggiornato. Il valore aggiornato è accessibile mediante args.value <pre>val =args.values[0];</pre>
change:	Ogni volta che la slider cambia di valore. Intercettato SOLO alla fine del movimento del cursore. Intercetta anche eventuali modifiche applicate via scripting.

Tutte le funzioni di evento hanno come parametro di args il **value** corrente del cursore

Metodi

value: Consente di impostare / leggere il valore corrente dello slider
values: si comporta come value ma viene utilizzato per slider con cursori multipli. Accetta come argomento l'indice (partendo da 0) del cursore su cui si vuole lavorare. Es:
`$("#selector").slider('values', 0, 25);`
 Assegna il valore 25 al primo cursore dello slider.

La classe spinner ()

Trasforma un generico tag DIV in uno Spinner, cioè un Text Box con i pulsantini di incremento / decremento.

Proprietà

min/max valori minimo e massimo impostabili. I valori possono essere anche negativi.
 Default null
step Passo di incremento / decremento. Default 1.
numberFormat: formato numerico: "n" numero decimale, "c" valuta, default null numero intero

La Property **value** non esiste ed è disponibile solo come metodo.
 Il valore dello spinner è inizializzato a 0.

Metodi

value Legge / Imposta il valore dello spinner

Eventi

spin: attivato soltanto in corrispondenza di ogni singolo click sui button up/down.
 All'interno dell'evento spin il **value** non è ancora aggiornato.
 Il valore aggiornato è comunque accessibile come **args.value**
change: attivato dopo ripetuti click solo nel momento in cui si preme invio o si sposta il focus.

Il metodo destroy()

```
if (spinner.data("ui-spinner" ))
    spinner.spinner("destroy");
else
    spinner.spinner();
```

Il metodo **destroy** rimuove le funzionalità della classe indicata e l'elemento ritorna alle sue origini HTML. Per testare se la classe è applicata oppure no su un certo elemento si utilizza il metodo **.data** passandogli il nome della classe con anteposto il prefisso **ui-**

Metodi statici di jQueryUI

\$.datepicker.setDefaults(\$.datepicker.regional['it']);
 Consente di impostare la lingua di apertura del datepicker.

Deferred and Promise

Si tratta di oggetti (molto simili) mirati a facilitare la gestione dei processi asincroni.

La programmazione asincrona interviene nel momento in cui si scrive del codice che blocca l'I/O per un lungo periodo di tempo. In questi casi tipicamente si indica al sistema qual'è l'operazione che dovrà essere eseguita al termine del codice "bloccante" e gli si passa una funzione detta di "callback", da richiamare ad operazione terminata. Il sistema si prende l'incarico di invocare la funzione di callback al momento opportuno e ritorna immediatamente il controllo,

Quando si hanno più operazioni da eseguire in cascata, in cui ogni operazione dipende dal risultato della precedente, occorre annidare le callback una dentro l'altra con estensione del codice verso destra creando quella che è conosciuta come "**Piramide della sventura**" (Pyramid of Doom).

L'oggetto Deferred ed il metodo statico \$.Deferred()

Il metodo statico `$.Deferred()` istanzia un nuovo oggetto **Deferred** poi richiama la funzione ricevuta come parametro alla quale inietta un puntatore **d** all'oggetto deferred medesimo, e infine restituisce al chiamante il medesimo puntatore **d** sotto forma di **Promise** (cioè prima di restituire l'oggetto esegue un cast da Deferred a Promise)

Esempio

```
_div1.show(3000, function(){  
    alert("ok");  
});
```

potrebbe essere riscritto nel modo seguente:

```
var promise = $.Deferred(function (d) {  
    _div1.show(3000, function() {  
        d.resolve();  
    });  
});  
  
promise.done( function() {  
    alert("ok");  
});
```

Cioè il codice della callback viene "portato fuori" rispetto alla funzione iniziale.

Molti metodi jQuery asincroni restituiscono al programmatore un oggetto **Promise**

- il metodo **d.resolve()** deve essere utilizzato in caso di successo.
- il metodo **d.reject()** deve essere utilizzato nel caso di errore
- il metodo **d.notify()** può essere utilizzato per segnalare che la funzione è in corso di esecuzione

L'evento **promise.done()** viene richiamato in corrispondenza del metodo **d.resolve()**

L'evento **promise.fail()** viene richiamato in corrispondenza del metodo **d.reject()**

L'evento **promise.progress()** viene richiamato in corrispondenza del metodo **d.notify()**

L'evento **promise.always()** accetta due funzioni di callback:

- la prima viene richiamata in corrispondenza del metodo **d.resolve()**,
- la seconda viene richiamata in corrispondenza del metodo **d.reject()**

L'evento `promise.then()` accetta tre funzioni di callback:

- la prima viene richiamato in corrispondenza del metodo `d.resolve()`,
- la seconda viene richiamato in corrispondenza del metodo `d.reject()`
- la terza viene richiamato in corrispondenza del metodo `d.notify()`

```
promise.then(  
    function(){ alert("ok")    },  
    function(){ alert("error")},  
    function(){ alert("in progress ....") }  
);
```

Differenza tra Promise e Deferred

L'oggetto **Promise** è sostanzialmente un oggetto **Deferred** di tipo read only, cioè che dispone soltanto degli eventi (**done**, **fail**, **progress**, **always**, **then**) ma NON dei metodi di risoluzione o rigetto del Deferred (`resolve`, `reject`, `notify`, etc)

Note

1) Ad ogni promise è possibile associare più funzioni di callback (eventualmente anche in cascata tra di loro) che verranno 'lanciate' sequenzialmente una dopo l'altra

```
promise.done(function(){ console.log(1); })  
    .done(function(){ console.log(2); })  
    .done(function(){ return $.get(...); })  
    // this executes after the $.get finishes.  
    .done(function(){ console.log(3); });
```

2) Gli eventi `.done()` e `.fail()` vengono eseguiti **anche** se la loro assegnazione viene eseguita *dopo* che le operazioni di `$.Deferred()` sono terminate (nel qual caso verranno eseguiti immediatamente).

3) Le funzioni di callback di `done()` e `fail()` possono ricevere uno o più parametri di qualunque tipo che dovranno essere impostati adeguatamente anche in fase di chiamata nei metodi `resolve()` e `reject()` e che, di conseguenza, verranno poi iniettati anche nelle callback dei metodi `done()` e `fail()`

Il metodo statico **\$.when()**

Il metodo statico `$.when(promise1, promise2, promise3)` è un metodo che viene eseguito solo dopo che **tutte** le promesse ricevute come parametro sono state terminate con esito positivo. Restituisce un oggetto promise al quale possono essere applicati i vari eventi precedenti.

Nel caso in cui una delle `resolve()` restituisca più parametri, questi verranno passati da `when()` alla promessa finale sotto forma di vettore

```
var promise1 = $.Deferred( function(d) {  
    d.resolve("Risultato promessa 1");  
});  
var promise2 = $.Deferred( function(d) {  
    d.resolve("Risultato promessa 2", "xx");  
});  
var promise3 = $.Deferred( function(d) {  
    d.resolve("Risultato promessa 3");  
});
```

```
$.when( promise1, promise2, promise3 ).done(
    function(risp1, risp2, risp3) {
        console.log(risp1 + "," + risp2[0] + "," + risp3); }
    );
```

Esempio 1 - Restituzione diretta di una promise

Anziché scrivere il codice all'interno di `$.Deferred()`, è molto più semplice utilizzare il puntatore restituito da `$.Deferred()`:

```
function visualizza (selector, speed) {
    var d = $.Deferred();
    var s = speed || "slow";
    $(selector).show(s, d.resolve);
    return d;
}

$.when(
    visualizza('#div1', 4000),
    visualizza('#div2', 3000)
).done(function () {
    alert('Animazioni terminate');
});
```

Le istanze `d` ritornate da `visualizza` sono utilizzate all'interno del metodo statico `$.when` per attendere che ogni promise sia terminata con successo.

Esempio 2 - Restituzione diretta di una promise

Qualunque funzione può essere resa asincrona nel semplicissimo modo seguente:

```
function foo() {
    return Promise.resolve(1);
}

foo().done(alert); // 1
```

Alla funzione `alert` viene automaticamente iniettato il valore 1 che verrà visualizzato.

Al posto del semplice valore 1 la funzione `foo` potrebbe passare a `resolve` il risultato di una elaborazione particolarmente pesante.

Esempio 3 : il metodo `.promise()`

Il metodo `.promise()` può essere applicato ad un qualunque oggetto della pagina html. Nel momento in cui viene applicato restituisce un oggetto `promise` contenente tutte le procedure di callback in quel momento accodate su quell'oggetto. Se la coda è vuota genera immediatamente l'evento `done`. Se la coda non è vuota genera l'evento `done` nel momento in cui la coda si svuota:

```
$("button").on("click", function() {
    $("p").append("Started...");
    $("div").each(function( i, ref ) {
        $(this).fadeIn().fadeOut( 1000 * (i+1) );
    });
    $("div").promise().done(function() {
        $("p").append(" Finished! ");
    });
});
```