Le librerie jQuery e jQueryUI

Rev Digitale 2.3 del 30/11/2017

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 …… 16

jQueryUI

Introduzione a jQuery User Interface …… 18
Animazioni sui colori …… 19
Animazioni sulle classi …… 19
Il metodo effect() …… 19

Le classi di interazione …… 20
Eventi e Proprietà …… 20
La gestione degli eventi e i parametri events e args …… 21
La gestione dei metodi …… 22
La classe resizable …… 22
La classe draggable …… 23
La classe droppable …… 24
La classe selectable …… 25
La classe sortable …… 26

I widget …… 26
button …… 27
controlgroup …… 27
autocomplete …… 27
dialog …… 28
accordion e tabs …… 29
datepicker …… 30
menù e sottomenù …… 31
progress Bar …… 32
slider …… 33
spinner …… 34

jQuery

The Write Less, Do More, JavaScript Library *(jquery.com).* jQuery è una libreria di funzioni (**framework**) Javascript, **cross-browser** che semplifica l'interazione con il DOM della pagina HTML, facilitando in particolare l'interazione tra javascript e le proprietà CSS. Pubblicato, 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)

1.9.1	04 febbraio 2013
2.1.4	ottobre 2015
3.1.1	ottobre 2016

Caratteristiche:

case sensitive proprietà, eventi e metodi hanno sempre la lettera iniziale **minuscola.**
apici singoli e apici doppi sono equivalenti. Per i **commenti** sono accettati sia /* */ sia //
punto e virgole finale è facoltativo, come in Java Script.

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

E' possibile specificare la versione a cui ci vuole collegare oppure scegliere la più recente.
jQuery è anche distribuito su diversi altri server fra cui google :

```
<script src="http://code.jquery.com/jquery-3.2.1.js">        </script>
<script src="http://code.jquery.com/jquery-latest.js">       </script>
<script src=" https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.js">   </script>
```

Il collegamento online ha il vantaggio che, se ho già visitato un sito che usa jQuery, il browser ha la libreria in cache ed evita di doverla ricaricare.

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 una collezione di tutti gli oggetti individuati,** questo 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(){});`

Come parametro è possibile passare anche combinazioni di selettori CSS che non sono supportate da un certo browser. jQuery: fornisce un livello di astrazione tale da azzerare le differenze tra i browser.

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 di callback da eseguire in corrispondenza dell'evento.
  Questa funzione **deve essere priva di parametri**

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

Il metodo **off** consente di rimuove un handler di evento.

```
$("div#wrapper").off("click");
```

In alternativa a `on` e `off` si possono utilizzare **bind** e **unbind** che presentano la stessa sintassi ma sono più pesanti in quanto consentono la gestione di un namespace completo.

Handler multipli

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

```
$("div#wrapper").on("click", funzione1);
$("div#wrapper").on("click", funzione2);
$("div#wrapper").off("click");  // li rimuove entrambi
```

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'unbind (che in caso di evento non associato risulta ininfluente) e *dopo* il bind:

Il metodo "**.one**" è simile a "**.on**" ma l'evento viene eseguito una sola volta anche in caso di bind multipli. In pratica al termine dell'evento rimuove se stesso.

Sintassi delle funzioni di evento

Tutti gli eventi jQuery si aspettano come parametro **un puntatore ad una funzione priva di parametri.** Questa funzione può essere scritta:

- In forma anonima direttamente all'interno dell'evento. Esempio:

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

  Ha il vantaggio di poter accedere a tutte le variabili esterne appartenenti al contesto in cui trova.

- Come funzione esterna di cui si passa il riferimento. Esempio:

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

```
}
```

nascondi' in realtà è una variabile di tipo reference che, in questo caso, contiene un riferimento a funzione. Questa sintassi ha il vantaggio di poter richiamare più volte la funzioni da ambiti differenti

in alternativa è anche possibile assegnare il riferimento a funzione direttamente ad una variabile e poi passare la variabile all'evento.

```
var nascondi = function () {      // riferimento a funzione
     $("div#wrapper").hide( );
}
$(#myButton).on("click", nascondi);
```

*Attenzione però che, usando questa sintassi, la variabile DEVE essere dichiara **prima** della procedura che la utilizza. Nei casi precedenti, invece, la procedura nascondi può essere dichiarata indifferentemente **prima** o **dopo** del suo utilizzo.*

Note sul metodo off( )

- Se si richiama il metodo off senza passare nessun parametro, questo **rimuove tutti i gestori di evento** associati all'oggetto ed assegnati tramite on. Es `$("p").off();`
- Se si richiama il metodo off passando il nome di un evento, vengono rimossi tutti i gestori di evento associati a quel singolo evento. Es `$("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),`
  dove il gestore di evento dovrà necessariamente essere stato espresso in forma esplicita.

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.
Questo collegamento diretto però "non consente la gestione dei delegated events".

Elenco dei principali metodi di evento

jQuery mette a disposizione un vasto elenco di metodi ed eventi che il programmatore può utilizzare. Tutti scritti completamente in minuscolo, senza camelCasing.

`.click()` - **`.dblclick()`**

`.change()` //cambio del contenuto del textBox (richiamato però solo quando si abbandona il campo)
`.select()`
`.mouseover()` -**`.mouseout()`** // Richamato nel momento in cui il cursore **entra** sull'elemento
`.mousemove()` // Richiamato durante il movimento del sensore sull'elemento
`.focusin()` // Richiamato nel momento in cui l'elemento riceve il focus
`.blur()` // Perdita del focus
`.keypress()` **`keyup()`** **`keydown()`**
`.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**().

.text( ); Restituisce il valore di un qualsiasi oggetto testuale restituito dal selettore.
.text("salve"); Assegna un valore ad un qualsiasi oggetto testuale

.html("salve"); Uguale a text, ma all'interno del testo accetta tag HTML (***preferibile***)
E' l'equivalente di innerHTML di javascript.
.html( ); Restituisce il valore di un qualsiasi oggetto testuale,
compresi eventuali tag html contenuti all'interno dell'elemento. Es:

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

.empty( ); Assegna un stringa vuota ad un qualsiasi oggetto testuale. (*innerHTML=""*)

.val( ); Restituisce il **value** dei controlli restituiti dal selettore corrente
.val("salve"); Assegna un valore all'attributo `value` di tutti i controlli restituiti dal selettore.
E' possibile passare valori multipli racchiudendoli fra quadre: (['uno', 'due'])

.width( ) e **.height( )** Consentono di leggere / scrivere direttamente width e height

.css("propertyName") Restituisce il valore di una singola CSS property del tag corrente
.css("propertyName", "valore") Consente di assegnare una CSS property al tag corrente
La property **deve** essere scritta sempre **CON** le virgolette, in formato css (es text-align) oppure senza virgolette in formato camelCasing. I valori alfanumerici **devono** essere scritta **CON** gli apici. I valori numerici **possono** essere scritti **CON** apici e UM oppure **senza** apici e UM

.css({propertyName:"valore", etc }) Sintassi alternativa per assegnazioni multiple

Per definire le proprietà CSS di un certo tag, è anche disponibile la seguente, comodissima, sintassi:

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

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

.addClass("nomeClasseSenzaPuntino") Applica una classe CSS all'elemento corrente
.removeClass("nomeClasseSenzaPuntino") Rimuove una classe CSS dall'elemento corrente
Entrambi accettano più nomi di classi separati da spazio
.hasClass("nomeClasseSenzaPuntino") Restituisce true se l'elemento implementa la classe
.toggleClass("nomeClasseSenzaPuntino") Aggiunge / Rimuove la classe

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 agli 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("AttrbuteName") Restituisce il valore di un attributo HTML del tag corrente,
.attr("AttrbuteName", "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("AttrbuteName") Restituisce il valore di un attributo HTML impostato dinamicam.

.prop ("AttrbuteName", "valore") Consente di assegnare un valore ad un attributo HTML. Es:

```
.prop("disabled", true);
.prop("disabled", false);
```

Con **prop** occorre utilizzare true/false.

Il metodo is()

Accetta come parametro un **selettore secondario.** Verifica se l'elemento corrente appartiene alla collezione identificata dal selettore secondario. **In pratica esegue un confronto fra i puntatori** L'esempio verifica se l'elemento corrente appartiene alla collezione degli elementi che implementano la classe "myClass".

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

Se l'elemento appartiene alla collezione secondaria restituisce `true`, altrimenti restituisce `false`. Se la collezione secondaria è vuota restituisce sempre `false`.

Effetti grafici

.show() Applicato ad un elemento nascosto, lo visualizza.

.hide() Applicato ad un elemento visualizzato, lo nasconde.
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.
Se ad una piccola porzione interna ad un tag DIV viene applicato il metodo `hide()`, quando il tag DIV verrà visualizzato la porzione nascosta continuerà a rimanere nascosta

.fadeIn () Applicato ad un elemento nascosto, lo visualizza.

.fadeOut () Applicato ad un elemento visualizzato, lo nasconde".
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.

.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 `l'hide()`, altrimenti esegue lo `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(),

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

1) duration indica la durata dell'animazione. Si possono utilizzare i valori `"fast" (200 ms)` e `"slow" (600 ms)` oppure un tempo espresso in msec Il default è 400ms. **Se non si specifica la duration, l'oggetto viene mostrato in un tempo di 400 ms, ma SENZA ANIMAZIONE.**

2) complete (secondo 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});
```

`.animate ( {width:"100px", etc}, mSec, 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. **NON consente invece l'animazione dei colori,** che sarà possibile solo con le librerie UI.
E' invece possibile animare la proprietà **`opacity`**: `$(ref).animate({opacity:0},1000);`
Gli stili impostati rimarranno attivi anche al termine dell'animazione, fino a esplicita rimozione.

`.disableSelection``()`; è un metodo di **jQueryUI** che, applicato ad un **wrapper**, inibisce la selezione del testo negli elementi interni a quello corrente. E' analogo alla property CSS3 **`user-select:none`**

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).

Se però la variazione del testo viene effettuata mentre l'elemento è nascosto, tale variazione risulta assolutamente impercettibile ed il testo verrà mostrato gradatamente tramite il fadeIn.

Viceversa se la variazione del testo avviene mentre il testo è visualizzato, la variazione sarà istantanea e SOLO DOPO partirà l'eventuale effetto di dissolvenza fadeOut.
Da un punto di vista grafico, è bene modificare il testo solo quando l'elemento risulta 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(msec)** richiamato fra due animazioni successive, consente di introdurre un ritardo fra le due animazioni. **Non è utilizzabile sul thread principale dell'applicazione**, ma solo sui thread secondari relativi alle animazioni. Es `$("mydiv").fadein(500).delay(500).fadeout(500);`

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

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 medesimo, per cui è possibile concatenare anch'essi come i normali metodi :

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

La collezione jQuey ed i puntatori JavaScript

- Il selettore $ restituisce **sempre** una **Collezione di oggetti del DOM** da intendersi sostanzialmente come una **Collezione di puntatori javascript**.
- La proprietà **.length** restituisce il numero di elementi appartenenti alla collezione.
- La collezione rimane tale anche quando l'elemento restituito è uno solo, nel qual caso sarà una collezione con lunghezza 1.
- **Alcuni metodi jQuery, ad esempio .css(), applicano il loro effetto a TUTTI gli elementi della collezione. Altri metodi jQuery, ad esempio .val(), applicano il loro effetto soltanto al primo elemento della collezione**

Si consideri il seguente esempio:

```
var _div = $("div");   // collezione di elementi div
alert(_div.length)
alert(_div.val());                  // valore del primo elemento
_div.css({backgroundColor:"red"})  // applicator a tutti gli elementi
alert(_div[2].value);
```

Il metodo **.val()** 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 java script e ne va a leggere il value

oppure:

```
$("p")[2].innerHTML = "Sono il terzo p della collezione jQUery";
```

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 del puntatore.

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

Il secondo esempio precedente potrebbe anche essere riscritto nel modo seguente:

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

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:

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

```
if(_div1==_div2)
   alert("i due elementi coincidono"); // sempre false
if(_div1[0]==_div2[0])
   alert("i due elementi coincidono"); // true
```

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 brower 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. alert($(event.target).attr("id"))
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.witch	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

```
function visualizza(event) {
    alert( "Hello " + event.data.name );
}
$("button").on("click", {name: "Karl"}, visualizza );
$("button").on("click",  "",  '{"name":"Karl"}',  visualizza );
```

Puntualizzazioni su jQuery

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(){
    $("#wrapper").text("salve mondo"); }
```

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. Anteponendo il dollaro si ottiene il corrispondente puntatore jQuery:

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

Visibilità delle variabili

Le variabili dichiarate all'interno dell'evento `$(document).ready()` saranno a visibilità globale per l'intero evento e quindi sostanzialmente per l'intera applicazione. Le variabili globali possono essere dichiarate indifferentemente all'interno o all'esterno dell'evento `ready`

Accesso alle variabili dalle funzioni di callback

Una funzione di callback scritta in loco **in forma anonima può accedere a tutte le variabili definite all'interno dell'oggetto in cui si trova**. Questo non è possibile se la funzione viene scritta esternamente

Inoltre **una funzione di callback può sempre accedere tramite il puntatore `this` all'oggetto che ha avviato la funzione di callback stessa**, sia nel caso delle funzioni scritte in loco**,** sia nel caso di 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){
  ```

Chiaramente la funzione di callback dovrà avere la firma definita all'interno del metodo che la usa. Quasi tutti i metodi jQuery richedono funzioni di callback prive di parametri.

Nel caso in cui occorra necessariamente passare dei parametri alla funzione di callback, la cosa più semplice è quella di scriverla in forma anonima **in loco** e, al suo interno, richiamare una funzione esterna passandogli i parametri desiderati.
Se invece la funzione di callback deve necessariamente essere scritta esternamente (ad esempio perché richiamata in più punti) la cosa più semplice è quella di dichiarare **globali** le variabili necessarie.

Richiamo sugli PseudoSelettori CSS

Sono introdotti dal simbolo **:** Possono essere utilizzati in due modi :

- **all'interno del selettore principale di accesso, nel qual caso consentono di filtrare gli elementi restituiti dal selettore**
- **all'interno del metodo .is() che rappresenta l'alternativa jQuery all'approccio precedente.**

In entrambi i casi restituiscono sostanzialmente true / false a seconda che la condizione sia vera o no.

```
input[type=radio]          // Equivalente a input:radio e :radio
input[type=chekbox]        // Equivalente a input:checkbox e :checkbox
input[type=text]           // Equivalente a input:text e :text
input[type=button]         // Equivalente a input:button e :button
input[type=submit]         // Equivalente a input:submit e :submit

input[name=txtNome]        // Vale solo per qualche attributo specifico e non per tutti gli attributi
input:radio[name=opt]
input:checkbox[name=chk]:checked

:enabled
:disabled
:checked
:selected
:visible
:hidden

:not(selettore_Secondario)  // Es :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 sottomenu (anche annidato)

$("select option[value=3]").attr('selected','selected');
$(":disabled").removeAttr("disabled");
```

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. Esempio:

```
<div id="wrapper">
      <p> …… </p>
      <p> <input ……> </p>
      <input ……>
      <p> …… </p>
</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 perchè, 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**)
`p:nth-child(i)`	// i-esimo figlio **generico** , se **di tipo p**
`p:nth-of-type(i)`	// i-esimo elemento **del suo tipo**, se **di tipo p** Se nth-child non è di tipo p, il selettore restituisce false
`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

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

Gli pseudoselettori **`:first`** e **`:last`** possono essere ulteriormente accodati a **`nth-child`** e **`nth-of-type`** per restituire primo ed ultimo elemento della collezione.
Sono analoghi ai metodi jQuery **`.first() e .last()`**

Navigazione della collezione restituita dal selettore (traversing)

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

`.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)` è simile a .eq( ) ma restituisce una collezione di elementi

`.each()` consente di scorrere tutti gli elementi restituiti dal selettore. E' sostanzialmente equivalente ad eq() ma realizza un ciclo **`for each`** invece di un ciclo **`for`** tradizionale.
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());
```

Esiste anche un metodo statico **`$.each()`** utilizzabile per scandire un semplice vettore.

I metodi each() sono sempre sincroni (bloccanti), nel senso che interrompono l'esecuzione fino a quando non sono terminati.

.filter(sel**)**
Consente di applicare dei filtri sulla collezione restituita dal selettore jQuery principale.

```
$( "li" ).filter(":even")
$( "li" ).filter(":even")
```

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

```
var voceSelezionata = optVoci.filter(':checked').val();
```

.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**. All'interno del selettore secondario è possibile utilizzare qualsiasi selettore / pseudoselettore CSS.

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)**;
Non vengono individuati i nipoti. Esempi:

```
$("#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")
```

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

.parent()
Il metodo parent() opera al contrario di children(), cioè restituisce il genitore dell'elemento corrente.

.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.

Vantaggi nell'uso dei metodi jQuery

Il vantaggio dell'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. Nell'esempio vengono inseriti nella collezione finale soltanto quei tag a che implementano la classe myClass

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

.not **(**sel**)**
Il metodo **not** restituisce solo quegli elementi che NON soddisfano alla condizione.

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

.siblings() restituisce TUTTI i fratelli del nodo corrente (nodo corrente escluso).

.next() .nextAll() .prev() .prevAll() .first() .last()

.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.

.first() seleziona il primo elemento di una lista. Analogo a :**first**
.last() seleziona l'ultimo elemento di una lista. Analogo a **:last**

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

.contents(sel)
è simile a `children` ed individua SOLO i figli di primo livello. La differenza sta nel fatto che individua anche eventuali nodi testuali (contenuto del tag). Inoltre consente di accedere anche ai contenuti di un IFrame.

.add(sel)
Consente di aggiungere un ulteriore gruppo di selezione a quello individuato dal selettore principale.
E' sostanzialmente analogo all'utilizzo della virgola all'interno del selettore CSS.

Note operative

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

```
$("div").text(Math.floor(10*Math.random())+1);  // stesso numero tra 1 e 10 per tutti i div
```

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

La **prima soluzione** assegna lo stesso numero a tutti gli elementi DIV della pagina.
La **seconda soluzione** assegna invece un numero specifico a ciascun DIV.

each() si usa quando occorre fare delle cose 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: Associazione di eventi all'interno di un ciclo

Se il bind di un evento viene eseguito all'interno di un ciclo for tradizionale del tipo

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

La variabile i sarà accessibile all'interno dell'evento `.on("click")` però verrà letto il valore di **`i`** in quel momento, cioè DIM (valore assunto dalla variabile i al termine del ciclo !). Si può ovviare a questo inconveniente salvando il valore di i all'interno di un attributo creato appositamente.

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

```
$("#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
   });
}
```

Nota 3: Creazione Dinamica

Supponendo di creare dinamicamente un insieme di pulsanti
`var btn= $("<button id="btn1"></button>");`

l'associazione dell'evento click deve essere fatta soltanto DOPO che l'oggetto è stato creato ed appeso al DOM. Cioè non è possibile scrivere ad un certo punto del document.ready una riga del tipo:

```
$("button").on("click", function(){})      // oppure
 $("#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 stato già creato ed appeso al DOM.

In alternativa di può associare l'evento attraverso la variabile stessa usata in fase di creazione dinamica:

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

In questo caso l'associazione funziona anche nel caso in cui l'elemento non sia ancora stato appeso al DOM.

Nota 4: Il metodo .val( ) nel caso di ListBox, Option Button e Radio Button

Utilizzo 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 vettore contenente i **value di tutte le voci selezionat**
Nel caso di radio button e check box restituisce il **value della voce corrente** (indipendentemente dal fatto che sia selezionata oppure no).

Utilizzo in scrittura

Nel caso del ListBox a selezione singola provoca la selezione dell'elemento indicato.
Nel caso del ListBox a selezione multipla, radioButton e chekBox, occorre necessariamente passare un **vettore di elementi** e provoca la selezione di tutti gli elementi indicati e l'automatica deselezione delle altre voci.

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

Creazione dinamica di nuovi elementi

append(ref**)**: Analogo a appendChild di javascript

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

Accetta come parametro sia il riferimento ad un oggetto jQuery (esistente o creato ex novo) sia una stringa html, che viene preventivamente convertita in oggetto e poi "appesa" al genitore.
Nel caso del parametro stringa, `.append()` diventa equivalente a `.html()`

```
var box = $("<div>Elemento <strong>nidificato</strong></div>");
```

Il $ iniziale istanzia in memoria un nuovo oggetto DIV e ne restituisce il puntatore.

```
blocco.addClass("nuovaClasse");
$("#contenitore").append(box);
```

Nel caso di elemento esistente, questo viene **tagliato** dalla posizione corrente ed **appeso** nella nuova posizione.

```
$("#menu1").append("#menu2 li:last-child");
```

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

```
var box = $("<div>Elemento <strong>nidificato</strong></div>");
box.addClass("nuovaClasse");
box.appendTo("#contenitore")
box.appendTo(document.body);   //inserisci nel body
```

Nel caso di elemento esistente, questo viene **tagliato** dalla posizione corrente ed **appeso** nella nuova posizione.

```
$("#menu1 li:last-child ").appendTo("#menu2");
```

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

Nel caso 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="categorie">
  <h1>Sport</h1>
  <div class="sub_cat">Calcio</div>
  <div class="sub_cat">Nuoto</div>
</div>
```

```
$(".sub_cat").before("<h2>Intestazione Sottocategorie</h2>");
```

produce il seguente risultato :

```
<div class="categorie">
  <h1>Sport</h1>
  <h2>Intestazione Sottocategorie</h2>
  <div class="sub_cat">Calcio</div>
  <h2>Intestazione Sottocategorie</h2>
  <div class="sub_cat">Nuoto</div>
```

Nel caso di elemento già esistente nella pagina, questo viene **tagliato** dalla posizione corrente ed incollato prima o dopo dell'elemento restituito dal selettore principale.

insertAfter(origin**)** e **insertBefore(**origin**)** operano in modo opposto rispetto ai due precedenti. **Tagliano** l'elemento restituito dal selettore principale e lo incollano prima o dopo dell'elemento passato come parametro.

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
```

wrapAll(): raggruppa tutti gli elementi individuati dal selettore e li avvolge con il nuovo tag
Accetta come parametro sia una stringa sia un riferimento jQuery o javaScript.

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

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

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

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

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

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, come ad esempio il numero di carte in esso contenute. La stessa cosa può essere fatta attraverso il nuovo metodo **.data()** applicabile anche agli elementi jQuery e non solo agli elementi della pagina html. Oppure anche mediante **.prop()**

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

All'interno di questi nuovo attributi è possibile anche salvare il puntatore ad un elemento jquery, potendo così poi rileggerlo in qualunque momento.

```
$("#viualizzatore").data("elementoVisualizzato", $(this));
```

La stessa cosa vale anche per i metodi **.css()** e **.addClass()**. Il primo consente di definire una nuova CSS property, ed il secondo una nuova classe (eventualmente anche non definita all'interno dei css) che può essere utilizzata per verificare se un elemento soddisfa o meno ad una certa condizione.

```
.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

`$.isArray($(selettore).val() )`
Restituisce TRUE se l'elemento ricevuto come parametro è di tipo Array

`$.inArray(stringa, vect)`
Restituisce l'indice della voce **`stringa`** all'interno del vettore **`vect`** (a base 0). -1 se non esiste.

`$.each(vect, function (index, elem) {`
Scandisce tutti gli elementi di un vettore. Index è l'indice a base 0, elem il contenuto.

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

`$.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();
```

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**.**

```
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js">  </script>
<script type="text/javascript" src="js/jquery-ui-1.10.3.custom.js">
```

All'interno dell'applicazione occorre specificare un link ai files **`jquery-ui.js`** e **`jquery-ui.css`**.
`jquery-ui.css` (*il cui uso è sconsigliato*) 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).

Effects

Elenco dei principali stili

ui-corner-all	generico, **spigoli arrotondati**
ui-widget-header	**intestazioni** Testo Bianco grassetto su IMG di sfondo arancione (più scuro sotto)
ui-widget-content	**corpo**. Testo #333 su IMG di sfondo grigia #eee height:100px, bordo #ddd;
ui-state-default	**stile applicato a tutti i pulsanti** (a, button, etc). Bordo grigio e Testo blu
ui-state-hover	mouse over.
ui-state-active	elemento attualmente attivo.
ui-state-disabled	elemento disabilitato.

Attenzione che quasi **tutte** 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 ( )`
- `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
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", {}, 900, function(){……});
```

Le classi di interazione

Le cosiddette classi di interazione, come ad esempio **`risizable()`**.**`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)

Queste opzioni possono essere di due tipi:

- **Proprietà** **`maxHeight:`**`250`
- **Eventi** **`start:`**`inizializza`

- **I valori numerici** DEVONO essere scritti in modo diretto **SENZA apici** e **SENZA unità di misura**.
- 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 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`**.

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

Le Property possono poi essere lette successivamente tramite la voce **`Option`** ed anche modificate:

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

Anche nel caso di proprietà a valore multiplo, all'interno di **Option** si scrive esattamente ciò che si scriverebbe all'interno della Property impostata direttamente.

La 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.
.which	For key or mouse events, indicates the specific key or button that was pressed.
.type	Describes the nature of the event.

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 possono anche essere richiamate esplicitamente come una normalissima funzione priva di parametri.

Definizione di un gestore di evento al 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 passere a **.on()** è dato dal nome della classe seguito dal nome dell'evento. Ad esempio nel caso dell'evento **start** della classe **resize** occorrerà scrivere::

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

Gestione dei Metodi: enable / disable / destroy / open / etc

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
```

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

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
cursor:"move"	Imposta la forma del cursore del mouse
containment: "#wrapper"	Consente il trascinamento **solo** all'interno del box indicato (per default all'interno del proprio contenitore)
cursorAt:{top:56,left:56}	Posizione del cursore durante il trascinamento
stack:	Fa si che gli elementi trascinati vengano trascinati e rilasciati SOPRA gli altri. `$("#wrapper div").draggable({stack:"#wrapper div"})`. Tutti gli elementi `div` contenuti dentro wrapper diventano trascinabili e, nel momento in cui vengono trascinati su un altro elemento dello stesso gruppo, vengano visualizzati al di sopra di esso
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
grid:[20,20]	Si muove "a step" di 20 pixel per volta.
scroll:true (default)	Se il genitore è overflow auto, quando si trascina un elemento contro i bordi del genitore, il genitore esegue uno scroll.
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 posiz 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.

helper:"clone" **original / clone** Impostando `clone`, non viene trascinato l'oggetto Draggable base, ma viene creata una copia (**helper**) e viene trascinata la copia. L'eventuale testo deve essere inserito in un apposito **tag** interno. Attenzione che la copia è un nuovo 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 bianco per cui sembra che venga trascinato soltanto il testo.
- Se si desidera che il clone trascinato abbia un aspetto diverso rispetto all'oggetto base, occorre assegnare a helper una funzione **custom**. Es:

```
helper:function(){
      var html=$("<div></div>");
      html.css({border:"1px solid black",
            width:"120px", height:"25px",
            backgroundColor:"#ffa",
            fontSize:"8pt",
            textAlign:"center" });
      html.text($(this).text());
      return html;
}
```

NB: Al momento del rilascio il clone scompare.

- **In caso di clone creato manualmente mediante la funzione precedente, si può appendere direttamente il clone all'interno del contenitore desiderato.**
- **In caso invece di clone creato mediante *helper:clone*, occorre eseguire un ulteriore clone del clone ed appendere il nuovo clone all'interno del contenitore desiderato:**

```
var ref = args.draggable.clone();    .
```

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

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 posizione corrente

Classi:

Gli elementi in corso di trascinamento implementano automaticamente la classe
`.ui-draggable-dragging`

I Metodi enable e disable

Dopo aver richiamato il metodo draggable per istanziare ed attivare il trascinamento, il trascinamento stesso può essere disabilitato / riabilitato richiamando ancora il metodo **`draggable`** con i valori **enable** o **disable.** I valori **enable** e **disable** possono essere passati a **`.draggable()`** soltanto **dopo** che la classe **`.draggable()`** è stata istanziata.

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 si che quell'oggetto possa intercettare il rilascio dell'elemento trascinabile se questo viene rilasciato al suo interno. In corrispondenza del rilascio viene scatenato il corrispondente evento **DROP**

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

Proprietà:

`accept: "Selector"` Accetta in DROP SOLO gli elementi identificati dal corrispondente selettore CSS. In alternativa è possibile assegnar una funzione che ritorni TRUE se l'elemento è accettato oppure FALSE in caso contrario:

```
accept: function() {
     if() return true;
      else return false;
},
```

`activeClass:"className"` Indica la classe che deve essere applicata all'elemento Droppable nel momento in cui è in corso il Drag di un elemento valido per il Drop sull'oggetto corrente

`hoverClass:"className"`	Indica la classe che deve essere applicata all'elemento Droppable nel momento un element Draggable "entra" sull'area dell'elemento Droppable

Eventi:

DROP = rilascio dell'oggetto

Nel caso dell'evento DROP il parametro **args** contiene le seguenti proprietà:

- **args.draggable** rappresenta un puntatore **jQuery** all'elemento draggable **dal quale ha avuto inizio il trascinamento**. (attenzione: puntatore jQUery a differenza di **event.target** che è un javascript)
- **args.helper** In caso di clone rappresenta un puntatore jQuery al clone in fase di trascinamento (ma solo nel caso di clone creato manualmente mediante una funzione utente, altrimenti args.draggable)
- **args.position** – posizione attuale dell'elemento trascinato sull'elmento droppable (relativamente alla pagina oppure relativamente all'elemento contenitore se questo è **position:relative**).
  Può essere usato anche in scrittura per modificare la posizione.

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.

Classi:

Agli elementi dichiarati selectable mediante il metodo **`selectabe`** vengono automaticamente associate 2 comodissime classi il cui contenuto (colori, immagini, etc.) non è definito all'interno degli UC-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 come ad esempio il colore di sfondo dell'oggetto, etc.
- utilizzate all'interno dell'evento stop per accedere, mediante un ciclo **`for-each`**, a tutti gli elementi selezionati.
- aggiunte manualmente da codice per eseguire delle selezioni direttamente da codice.

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

Nota: 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 :*

- *la seleziona multipla è possibile solo più tramite il rettangolo. Il click è invece intercettato come inizio trascinamento.*
- *non è possibile trascinare in gruppo tutti gli elementi selezionati.*
  In corrispondenza del click viene trascinato soltanto l'elemento selezionato.

Per trascinare più oggetti contemporaneamente occorre inserirli in un contenitore e poi applicare il draggable al contenitore, però in questo caso il contenitore non può più essere selectable.

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 una serie di opzioni di tipo Proprietà / Evento.

La classe button( )

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

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

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

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

icon: "ui-icon-newwin"

L'opzione **icon** consente di visualizzare una icona a sinistra oppure a destra del testo del pulsnate. 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-specifica,** 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). Attenzione che le icone sono applicabili solo al tag *button* e non al tag <*input type="button*">

iconPosition: "end"

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

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). Se si desidera cambiare il colore di sfondo ai pulsanti premuti, è sufficiente ridefinire questa classe eliminando l'immagine di sfondo ed impostando il colore desiderato.

La classe controlgroup( )

Applicato ad un **contenitore** contenente un insieme di controlli, applica una trasformazione grafica ai controlli contenuti in quel contenitore. Nel caso di RadioButton che nel caso dei CheckBox l'effetto grafico risultante è identico. Nella versione **1.12** oltre a **Controlgroup** è stato aggiunto un nuovo widget **Checkboxradio** che però non deve essere applicato ad un contenitore ma ai controlli interni.

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",          // omettere px !!
     height:"300",         // omettere px !!
     autoOpen:false,
     show:"slow",
     hide:"explode",
});
```

Proprietà:

`autoOpen:``false` Il dialog box, per default, viene aperto immediatamente in fase di stanza. 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 widget.

`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 in corrispondenza dell'apertura e della chiusura della finestra. Si possono specificare un **`effetto`**, oppure una **`durata`**, oppure **`entrambi`**. 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 esplicta 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 si 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 DIV principale deve fungere da Panel per l'intero controllo.
Un tag UL interno fungerà da barra dei menu, in cui ogni voce conterrà un link al DIV corrispondente
Ogni scheda fisica sarà rappresentata da un tag DIV interno al Panel.

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

Proprietà

active: indice della scheda visualizzata (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 ( )

Apre 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 dataPicker 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 impstato 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. Imposatndo 1,1 viene mostrato un solo mese. Impostando 1,3 vengono mostrati 3 mesi su 3 colonne successiva 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 munù in un vero e proprio menù.

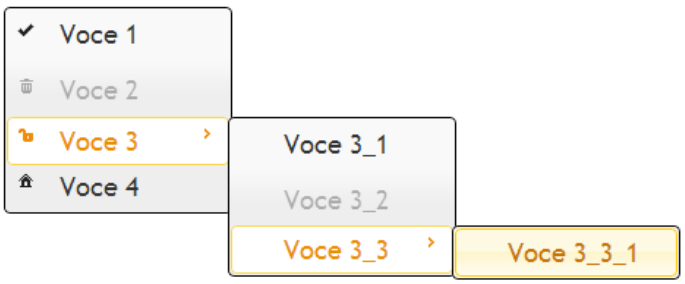
```
$("#menu").menu({
     select: visualizza,
     position: { my: "left top", at: "right-10 top+10" }
});
```

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 immagina 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'efffetto 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)

```
var values = $("#selector").slider( "option", "values" );
$("#selector").slider( "option", "values", [ 10, 25 ] );
```

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**

```
val =args.values[0];
```

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.