

# Bibbia FPW

## HTTP

HTTP è il protocollo utilizzato dai server web per rispondere alle richieste dei client.

Inizialmente era supportato solo dai browser, ma ora anche da altri applicativi.

La versione attuale è HTTP/1.1. HTTP/2 esiste ma è in fase di sviluppo.

Sta per **hypertext** (documenti testuali) **transfer** (trasmessi da remoto) **protocol** (grazie ad un protocollo).

Funzionamento: ad ogni richiesta del client corrisponde una richiesta del server. Il protocollo non ha stato, in quanto ogni risposta non dipende dalle precedenti.

Le richieste viaggiano su un protocollo di trasporto (TCP). Possono essere inviate più richieste nella stessa connessione.

### Richiesta HTTP:

Risorsa: identifica il contenuto da richiedere al server ed è identificato da un URI (uniform resource identifier).

Metodo: i dati o le richieste al server possono essere inviati in questi due metodi:

- GET: il client richiede i dati mettendoli nell'url;
- POST: i dati inviati sono contenuti nel payload;
- HEAD: funziona come get ma il server risponde solo con header (permettono di specificare delle preferenze di comunicazione).

Codici di stato:

- Da 200 a 299 successo;
- Da 300 a 399 reindirizzamento;
- Da 400 a 499 errore da parte del client (noi);
- Da 500 a 599 errore da parte del server.

Cookies:

Sono delle coppie chiave valore contenute fra più richieste e risposte fra client e server. Il server crea il cookie e lo invia al browser. Il browser lo salva in modo persistente e lo associa al dominio. Alla richiesta successiva il client prende il cookie e lo rimanda al server con un header cookie, per evitare per esempio di eseguire di nuovo il login.

## Uniform Resource Locator (URL)

```
<protocollo>://<nomehost>:<porta><percorso>?<querystring>#<fragment>
```

- Il **protocollo** applicativo da utilizzare per reperire la risorsa descritta (HTTP, HTTPS, FTP, MMS)
- Il **nome host** è costituito o da un nome di dominio o direttamente da un indirizzo IP
- La **porta** viene specificata quando il protocollo non utilizza quella standard (opzionale). Se presente è preceduta da :
- Il **percorso** è il path locale della risorsa richiesta all'interno del server
- La **query string** è una serie di coppie nome-valore con il formato `<nome1>=<valore1>&... &<nomeN>=<valoreN>`, che sono usati dal server come parametri. Inizia con un ?
- Il **fragment** viene utilizzato per specificare una posizione particolare all'interno della risorsa. Inizia con un #

```
http://fpw.unica.it:80/corso.jsp?anno=2021&lezione=mvc#conclusioni
```

\*EMOJI CHE ESPLODE\*

## HTML (HyperText Markup language)

È un linguaggio di marcatori che descrive la struttura di un documento utilizzando un insieme di tag.

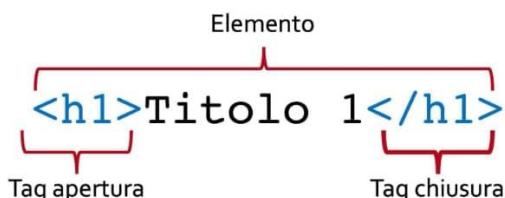
Questo mix di codice e testo forma una pagina web.

Il browser disegna il documento interpretando i tag.

NON deve essere utilizzato per lo stile.

I tag sono contenuti tra parentesi angolari. I tag sono composti da tag di apertura ed eventualmente anche di chiusura, nei quali si scrive uno slash prima della parola chiave.

Un elemento è tutto ciò che è contenuto tra un tag di apertura e uno di chiusura, compresi essi.



Gli elementi HTML possono avere degli attributi che forniscono informazioni addizionali su di essi, sono specificati nel tag di apertura, e sono formati da coppie nome-valore (i valori vanno sempre messi tra virgolette).



## Struttura di una pagina HTML

```
<!DOCTYPE html>

<html>

    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=utf-8">
        <title>Page Title</title>
    </head>

    <body>
        . . .
    </body>

</html>
```

La struttura nidificata degli elementi crea un albero.

Il tag `<!DOCTYPE ...>` definisce quale versione dell'HTML vogliamo usare.

I commenti si fanno così: <!--missili-->

Il tag `<head>` contiene elementi con informazioni generali sulla pagina tra i quali:

- Titolo;
  - Fogli di stile;
  - Inclusione di script;
  - Metadati (informazioni sul documento).

<title> definisce il titolo del documento e ciò che apparirà nella finestra del browser.

`<base>` permette di ridefinire l'URL di base per tutti i collegamenti relativi alla pagina.

<link> collega il documento a una risorsa esterna.

<body> racchiude la struttura principale del documento.

**Titoli:** i tag <h1>, <h2> sono preimpostazioni per i vari titoli della pagina. Sono considerati dal documento dal più importante al meno importante. Anche i motori di ricerca li usano per capire la struttura del documento, non solo il browser.

I paragrafi si implementano col tag <p>. I browser aggiungono una linea vuota prima e dopo il testo contenuto nel tag.

**Tag di formattazione:** servono per formattare il testo. Ci sono:

- <strong> e <b> per il testo in grassetto (dove strong è anche semantico e indica un testo più importante);
- <i> per il testo in corsivo;
- <em> per il testo con enfasi;
- <code> per il codice;
- <sub> e <sup> per pedice e apice;
- E altri (mark, small, del, ins).

**Link:**

Parole o immagini che se cliccate portano ad un altro documento o una sua parte (link interni). Il tag è <a>, usato insieme all'attributo “href” per indicare il collegamento.

**Immagini:**

Definite dal tag <img>, sono risorse esterne al documento e collegate ad esso tramite l'attributo “src” (source). Si possono specificare altezza, larghezza, bordi e tanto altro.

**Tabelle:**

Definite dal tag <table>, e hanno dei sottotag: <th> per le intestazioni, <tr> per le righe e <td> per le celle. Non si usano per costruire il layout.

### **Liste:**

Possono essere di due tipi: ordinate (`<ol>`, `enumerate`) o non ordinate(`<ul>`, `non enumerate`). Gli elementi all'interno si taggano con `<li>`. Anche loro hanno stili css dedicati.

### **Form:**

Contiene elementi che consentono l'input da parte dell'utente (campi di testo, caselle, pulsanti ecc). Sono utilizzati per inviare dati ai server con GET o POST (scelto con l'attributo "method") e hanno un attributo chiamato "action" specificante dove andranno i dati.

### **Input:**

Insieme di elementi utilizzati per ricevere un valore dall'utente, sono contenuti nei form con il tag `<input>`. Sono differenziati dall'attributo "type". L'input viene fornito come coppia chiave-valore, la chiave è specificata dall'attributo "name" e il valore nell'attributo "value". È possibile specificare una etichetta con il tag `<label>`.

#### **Input testuale:**

Serve per raccogliere testi inseriti dall'utente. Grazie all'attributo "type" si può specificare il tipo di testo (testi brevi, lunghi, password, email, date, numeri).

#### **Pulsanti:**

Si possono inserire con due tag diversi: `<input>` e `<button>`. Entrambi ammettono i seguenti valori per l'attributo "type": submit (invia un form al server), button (bottone generico usato con javascript) e reset (cancella gli input dell'utente dai campi della form). Input accetta solo l'attributo value, mentre button può contenere elementi figli (come immagini o testo). Input può avere una immagine come etichetta grazie all'attributo "type".

## Selezione a scelta singola

```
<h2>Cardinalità bassa</h2>
<p>
    <label for="man">Uomo</label>
    <input type="radio" name="sesto"
        id="man" value="uomo"><br>
    <label for="woman">Donna</label>
    <input type="radio" name="sesto"
        id="woman" value="donna"><br>
</p>

<h2>Cardinalità media</h2>
<p>
    <select name="marca">
        <option value="fiat">Fiat</option>
        <option value="alfa">Alfa Romeo</option>
        <option value="opel">Opel</option>
        <option value="audi">Audi</option>
    </select>
</p>

<h2>Cardinalità alta</h2>
<p>
    <select name="lista" size="4">
        <option value="1">Valore 1</option>
        <option value="2">Valore 2</option>
        <option value="3">Valore 3</option>
        <option value="4">Valore 4</option>
    </select>

```

### Cardinalità bassa

Uomo   
Donna

### Cardinalità media

Fiat

### Cardinalità alta

Valore 1  
Valore 2  
Valore 3  
Valore 4

## Selezione a scelta multipla

```
<h2>Cardinalità medio-bassa</h2>
<p>
    <input type="checkbox" name="vehicle" value="Bike">
    Ho una bicicletta
    <br>
    <input type="checkbox" name="vehicle" value="Car">
    Ho un'automobile
    <br>
    <input type="checkbox" name="vehicle" value="Moto">
    Ho una moto
    <br>
</p>

<h2>Cardinalità medio-alta</h2>
<p>
    <select name="lista" size="4" multiple>
        <option value="1">Valore 1</option>
        <option value="2">Valore 2</option>
        <option value="3">Valore 3</option>
        <option value="4">Valore 4</option>
    </select>
</p>
```

### Cardinalità medio-bassa

Ho una bicicletta  
 Ho un'automobile  
 Ho una moto

### Cardinalità medio-alta

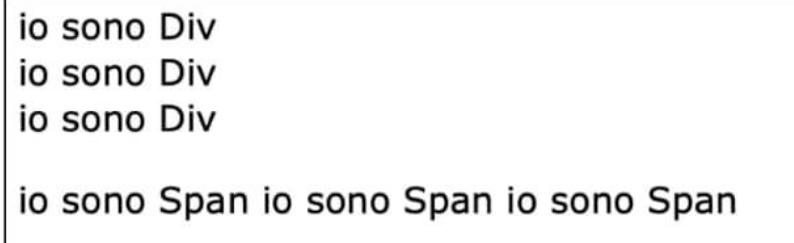
Valore 1  
Valore 2  
Valore 3  
Valore 4

## Sezioni dei documenti HTML:

Vengono utilizzate per separare i contenuti e definire il layout. Hanno due elementi:

- <div>, definisce un blocco separato dagli altri;
- <span>, definisce un blocco inline, non separato dagli altri.

L'immagine mostra la differenza tra creare tre div di fila e tre span di fila:



io sono Div  
io sono Div  
io sono Div

io sono Span io sono Span io sono Span

## Sezioni semantiche:

Sono sezioni specifiche per parti di testo o componenti di una pagina. Dal punto di vista visivo non presentano alcuna differenza con una sezione generica, ma hanno una semantica e se usate correttamente aiutano il browser a comprendere il documento. I tag semanticici sono stati introdotti con HTML 5 e rendono molto più semplice specificare la struttura del documento rispetto ad HTML 4.

Esempi importanti sono:

- 
- Header;
  - Nav;
  - Article, con al suo interno section e aside;
  - Footer.

### **Canvas:**

Permette di definire una sezione in cui poter disegnare grafica 2D, utile per applicazioni come i giochi.

### **Drag and Drop:**

Anche questo introdotto con HTML 5, permette di trascinare file esterni dentro una pagina web, un esempio è il caricamento di allegati su gmail. Utilizza il tag <div> con dentro gli attributi “ondrop” e “ondragover”.

Video e Audio:

Si possono includere con i tag <audio> e <video>.

### **Search Engine Optimization (SEO):**

Ottimizzazione per i motori di ricerca, viene effettuata per migliorare il posizionamento nelle SERP (Search Engine Result Page).

I motori di ricerca scansionano automaticamente e continuamente le pagine del web seguendo i link alla ricerca di nuovi contenuti.

Indicizzano poi questi contenuti e li memorizzano nei loro server.

Quando l'utente effettua una ricerca gli algoritmi del motore di ricerca creano la lista dei risultati in base all'indicizzazione dei contenuti sul server, e creano un ranking dei siti che utilizzano per mostrare all'utente i siti correlati alla sua ricerca.

L'ottimizzazione SEO va gestita in tutte le parti che compongono l'applicazione web. Alcuni accorgimenti si possono effettuare a livello pagina HTML:

- Inserire il tag <title> con un titolo breve e conciso;
- Inserire sempre il tag <meta name="description" ...> che contenga una descrizione informativa e interessante;
- Inserire il tag <meta name="keywords" ...> che contenga delle parole chiavi pertinenti;
- Non creare troppe ramificazioni ed evitare link che portino a pagine inesistenti;
- Usare nel tag <img> l'attributo “alt” con la descrizione breve e significativa dell'immagine e l'attributo “src” con un nome di file breve e descrittivo;
- Non utilizzare più di un tag <h1> nella stessa pagina;
- Non abusare di <strong>;
- Creare degli url semplici da leggere.

# CSS (Cascading Style Sheets)

Definiscono come visualizzare gli elementi HTML, evitando di fare altro nel file HTML se non definire la struttura del documento. Essendo esterni ad esso possono essere utilizzati da più documenti insieme e permettono una manutenzione più facile.

Il CSS è formato da un insieme di regole. Ogni regola seleziona un insieme di elementi HTML (per id, classe o altro), e agli elementi selezionati viene applicato il corpo della regola (la dimensione o il colore del testo, i bordi ecc.).

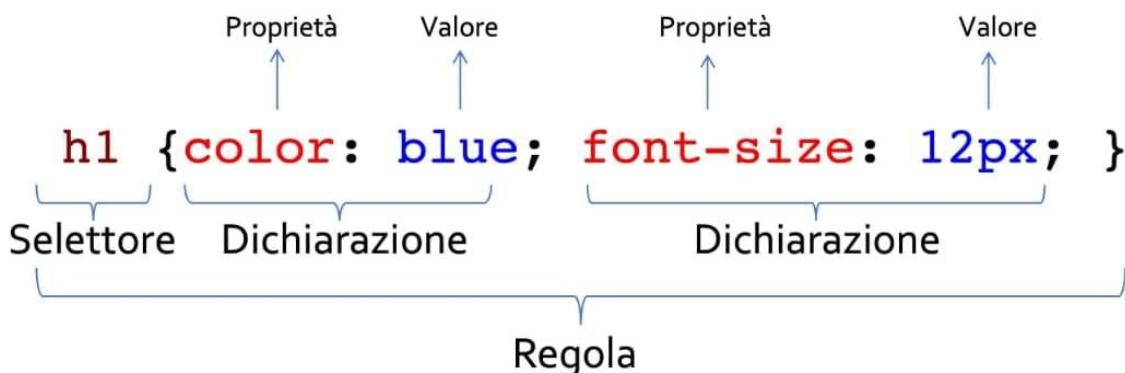
I CSS si includono con

- Tag `<link>` per utilizzare file esterni (consigliato)

```
<head>
  <link rel="stylesheet" type="text/css" href="mystyle.css"
        media="screen">
</head>
```

- Tag `<style>` per specificarli all'interno del file HTML.

L'attributo "media" permette di specificare la tipologia di dispositivo (pc, smartphone ecc.).



Il selettore specifica per quali elementi è valida la regola, la quale contiene una lista di dichiarazioni, le quali specificano il valore di un particolare attributo di stile (es: colore).

La proprietà indica il nome dell'attributo di stile e il valore come esso deve apparire.

I commenti sono come in C e Java.

Selettori:

Vengono utilizzati per selezionare un insieme di elementi dall'albero del documento.

Ne esistono di diverse categorie: universali, tipi (tag), classi e identificatori.

Alcuni sono basati sulle relazioni dell'albero HTML e altri su condizioni specifiche.

- Quelli universali selezionano qualsiasi elemento all'interno del documento e si indicano con \*;
- Tipi: selezionano tutti gli elementi di un determinato tipo (tag), per esempio h2;
- ID: indicato dal valore seguito da un cancelletto (#) seleziona l'elemento che ha l'attributo "id" (unico) uguale ad esso;
- Classi: come per l'id ma con il punto (.), ed è collegato all'attributo "class". A differenza dell'id un elemento può appartenere a classi diverse in questo modo:

```
<h2 class="cl1 cl2"> ... </h2>
```

- Combinazione tipo-id: seleziona solo gli elementi di un dato tipo con un determinato id (si specifica prima il tipo e poi l'id con #);
- Combinazione tipo-classe: come tipo-id ma con la classe. Se si specificano più classi l'elemento è selezionato solo se appartiene a tutte;
- Relazioni su albero: discendenti: permette di selezionare gli elementi di un certo tipo che siano discendenti da un altro (es: h1 em{...});
- Relazioni su albero: figli: permette di selezionare tutti gli elementi di un certo tipo figli di un altro con > tra i due tipi (prima il padre);
- Condizioni su attributi: selezione in base ai valori di un attributo. Esempio:

HTML	a[rel="copyright"] { color: blue; }	a[rel~="copyright"] { color: blue; }
<a rel="copyright"> ... </a>	✓	✓
<a rel="copyright copyleft"> ... </a>	✗	✓
<a rel="copyeditor"> ... </a>	✗	✗

- Pseudo-classi: sono utilizzate per selezionare elementi con informazioni che stanno fuori dalla definizione del documento. Esempi più importanti:

- **a:link** definisce gli stili per i link non visitati
  - **a:visited** definisce gli stili per i link visitati
  - **img:hover** definisce gli stili per un elemento quando il mouse ci passa sopra (senza click)
  - **button:active** definisce gli stili per un elemento quando viene attivato (es: click di un bottone)
  - **input:focus** definisce gli stili per un elemento di input quando viene selezionato (focus)
- Pseudo-elementi: elementi senza semantica che permettono di selezionare elementi generici che si trovano vicini ad altri, per esempio è possibile cambiare ciò che sta prima o dopo un elemento di tipo <h1> attraverso h1::before o h1::after. Hanno due principali vantaggi: non appesantiscono il documento con contenuto senza semantica e permettono di decidere cosa inserire di decorativo a seconda del dispositivo.

Si possono raggruppare più selettori grazie ad una lista separata da virgole. In questo modo si concentra la definizione della regola e si evita di scrivere codice ridondante:

```
.c11, .c12 {color: blue; font-size: 12px;}
```

```
.c11 {color: blue; font-size: 12px;}
.c12 {color: blue; font-size: 12px;}
```

## Esempi particolari

- **.c11** è equivalente a **\*.c11**
- **#id1** è equivalente a **\*#id1**
- I tag **div** con classi **c11** e **c12** si selezionano con **div.c11.c12**
- I tag **div** con classi **c11** o **c12** si selezionano con **div.c11, div.c12**
- **div .c11 .c12**  
Seleziona tutti i gli elementi con classe **c12** discendenti di un elemento con classe **c11**, a sua volta discendente da un **div**
- **div.c11 .c12**  
Seleziona tutti i discendenti con classe **c12** di tutti i **div** che abbiano classe **c11**
- **div .c11.c12**  
Seleziona tutti gli elementi con classe **c11** e **c12** discendenti da un **div**
- Esempio selettore con condizione su attributo combinato con pseudoclasse **a[title="text"]:link** (**:link** indica i link non visitati)
- **a:visited:hover** seleziona i link visitati quando ci si passa sopra col mouse

## Gli Stili:

Si possono specificare tante cose col CSS! 😊

### Dimensioni:

Si possono utilizzare vari modi per definire le dimensioni degli elementi:

Unità	Descrizione
%	percentuale (rispetto alla dimensione del contenitore)
in	pollici
cm	centimetri
mm	millimetri
em	1em è la dimensione del font corrente. Viene utilizzato per specificare dimensioni di testi che scalano in relazione alle preferenze del lettore. Si prende come unità il font corrente e poi si specificano multipli e sottomultipli per gli altri (es. 2em per i più grandi e 0.8 em per i più piccoli)
pt	Un punto è 1/72 di pollice (tipografico)
px	Un pixel è un punto sullo schermo del computer

### Colori:

Si possono definire tramite testo, valori RGB (in esadecimale (#FF0000) o base 10 con i tre canali (rgb(255,0,0))), e le trasparenze vengono gestite con l'attributo “opacity”.

### Sfondo:

Il colore si definisce attraverso la proprietà “background-color”. Es:

```
body {background-color: #FFFFFF}.
```

È anche possibile usare una immagine come sfondo, tramite “background-image”:

```
body {background-image: url(sans.gif)}.
```

È possibile bloccare la ripetizione su uno dei due assi (o entrambi) tramite la proprietà background-repeat, mentre con “background-position” si stabilisce dove posizionare l’immagine se non è ripetuta.

## **Testo:**

Gli attributi per definire le proprietà del testo sono:

- color: il colore;
- text-align: l'allineamento;
- text-decoration: la decorazione;
- text-indent: l'indentazione in pixel;
- font-family: il font, che può essere di diverse tipologie (come serif, sans serif ecc.);
- font-style: lo stile del font (corsivo o normale);
- font-size: la dimensione;
- font-weight: lo spessore.

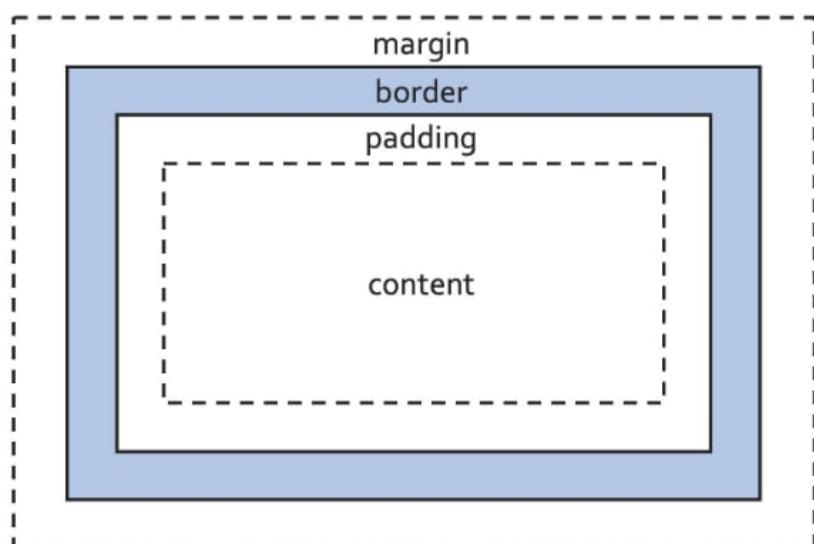
Si possono utilizzare font diversi da quelli di default tramite @font-face o <link>, con il quale possiamo aggiungere font in vari formati. Si possono aggiungere online o in locale.

## **Liste:**

Attraverso la proprietà list-style-type è possibile controllare i pallini o i numeri delle liste. Per le liste non ordinate vi sono: none, circle, disc, square; per quelle ordinate si hanno: decimal, lower-alpha, lower-roman, upper-alpha, upper-roman. È possibile anche impostare delle immagini come bullet scrivendo list-style-image.

## **Box model:**

Per il browser ogni elemento è un rettangolo. In questo modo:



Content è il contenuto dell'area vero e proprio, padding è lo spazio tra il bordo e il contenuto, border è un bordo che racchiude il rettangolo e margin è lo spazio libero intorno al bordo.



Si possono controllare le dimensioni di tutti i lati impostando un solo valore, altrimenti grazie alle proprietà -top, -right, -bottom, -left si possono controllare i lati uno per uno.

Le dimensioni del content si controllano tramite due proprietà: width e height. Se non specificate vengono calcolate dal browser.

L'altezza e la larghezza totali dell'elemento sono date dalla somma delle sue parti (margin, border, height, ecc...).

**none**: Nessun bordo.



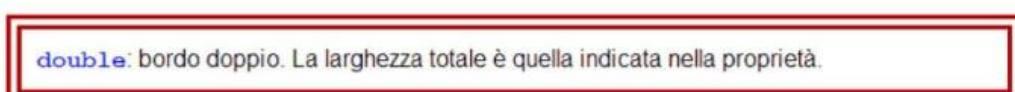
**dotted**: bordo con tratteggio a punti.



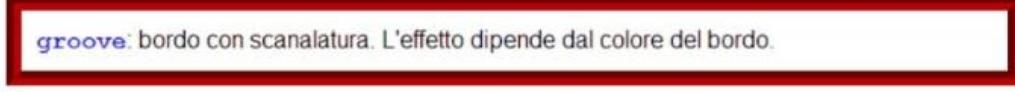
**dashed**: bordo con tratteggiato.



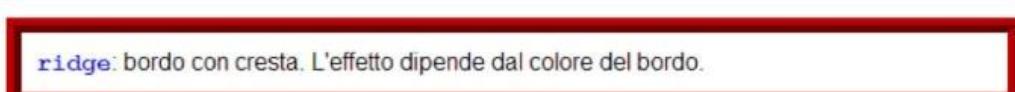
**solid**: bordo continuo.



**double**: bordo doppio. La larghezza totale è quella indicata nella proprietà.



**groove**: bordo con scanalatura. L'effetto dipende dal colore del bordo.



**ridge**: bordo con cresta. L'effetto dipende dal colore del bordo.



**inset**: bordo incastonato (interno). L'effetto dipende dal colore del bordo.



**outset**: bordo incastonato (esterno). L'effetto dipende dal colore del bordo.

## Tabelle:

Si impostano le dimensioni delle celle come per i normali box; avendo ogni cella un bordo di norma vengono “doppi”. Per questo si fa così:

**border-collapse: collapse**

--	--

**border-collapse: collapse**

--	--

È buona norma evidenziare le intestazioni con stili diversi da quelli delle altre righe e utilizzare colori di sfondo alternati per esse.

Titolo	Anno	Stagioni
How to Get Away with Murder	2014	6
La Casa di Carta	2017	3

## Display e visibility:

*display* regola il modo in cui un elemento viene visualizzato:

- **block** l'elemento prende tutta la larghezza disponibile e viene aggiunta un'interruzione di riga prima e dopo di lui
- **inline** prende solo la larghezza disponibile e non si aggiungono le interruzioni di riga prima e dopo
- **none** nasconde l'elemento, facendo in modo che non prenda nessuno spazio

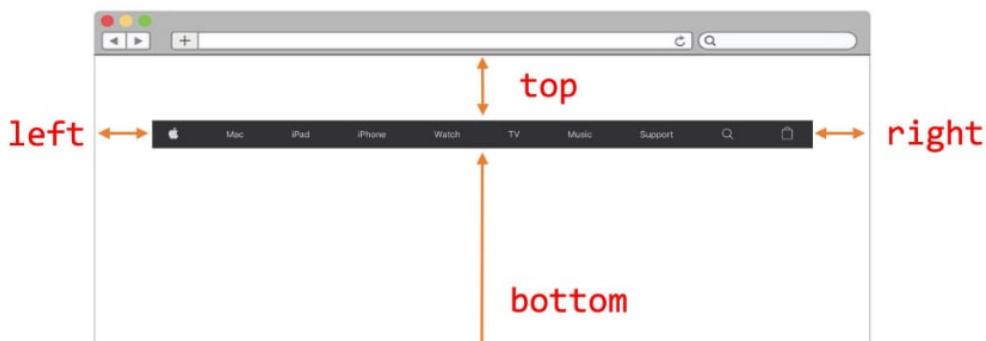
La modalità di default del *display* dipende dell'elemento. Un elemento può essere nascosto anche impostando *visibility: hidden* lasciando lo spazio.

Altri valori:

- *inline-block*: elemento inline del quale si può controllare altezza e larghezza;
- *table table-row table-cell*: dispongono gli elementi come se fossero dentro una tabella ma senza utilizzare le tabelle.

## Posizionamento:

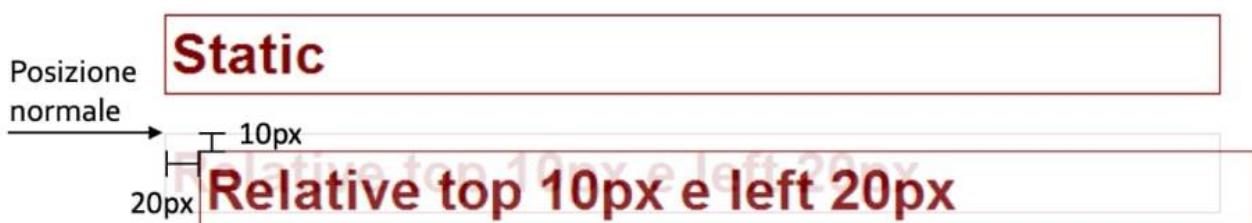
È controllato dalla proprietà *position*.



- *static*: gli elementi vengono posizionati secondo il flusso normale della pagina, è il valore di default.

- *fixed*: l'elemento è posizionato in modo relativo alla finestra del browser, quindi rimane nella stessa posizione anche se si scorre la pagina.

- *relative*: consente, tramite le proprietà *top*, *bottom*, *left* e *right*, di modificare la posizione dell'elemento partendo dalla sua posizione normale nella pagina.



- *absolute*: consente tramite *top* *bottom* *left* *right* di posizionare un elemento rispetto alla posizione del primo elemento che lo contiene che abbia un valore di posizione diverso da *static*. Se questo elemento non si trova si considera il blocco HTML.

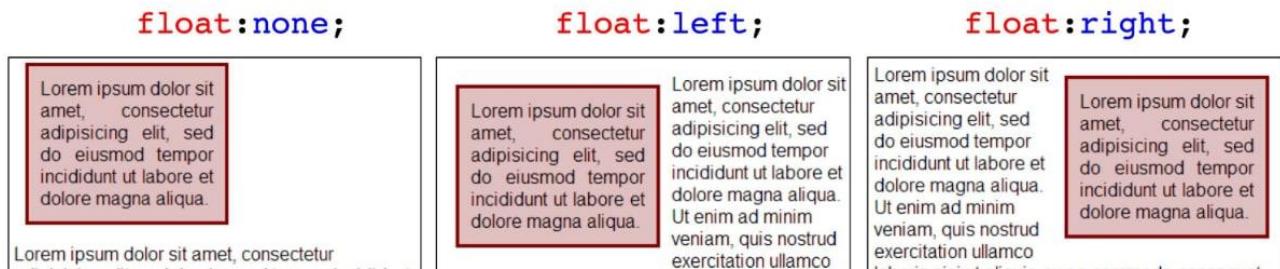
## Sovrapporre elementi:

Modificando i valori degli attributi di posizionamento è possibile sovrapporre degli elementi.

*z-index*: permette di controllare quale elemento sia in primo piano e di assegnare un valore ad ogni elemento disegnando sotto quelli con valori più bassi e sopra quelli con valori più alti.

## Float:

Permette di accorpare gli elementi orizzontalmente facendo in modo che vengano come “avvolti” dal resto della pagina.



È possibile addossare più di un elemento. Il browser li addosserà finché c'è spazio e poi andrà a capo.

La proprietà *clear* permette di bloccare l'addossamento a destra, a sinistra, su entrambi i lati o da nessuna parte.

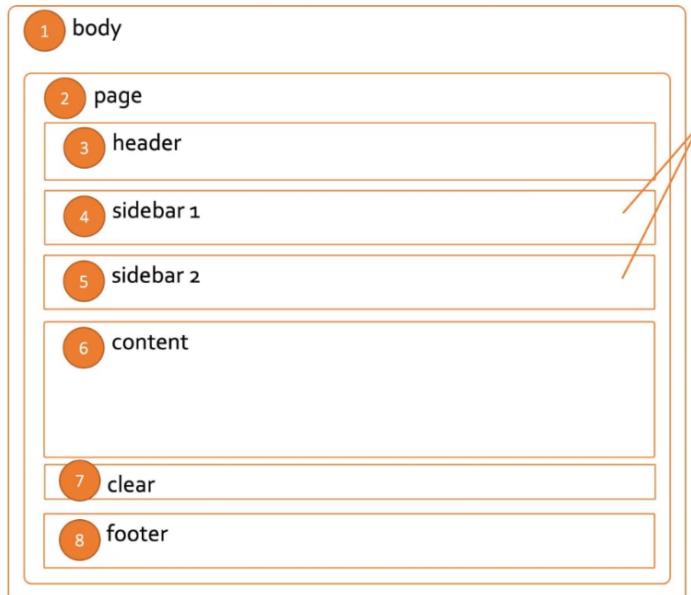
## Layout patterns:

Ci sono di tipi di design, ossia fluido e a larghezza fissa, ognuno con tre varianti:

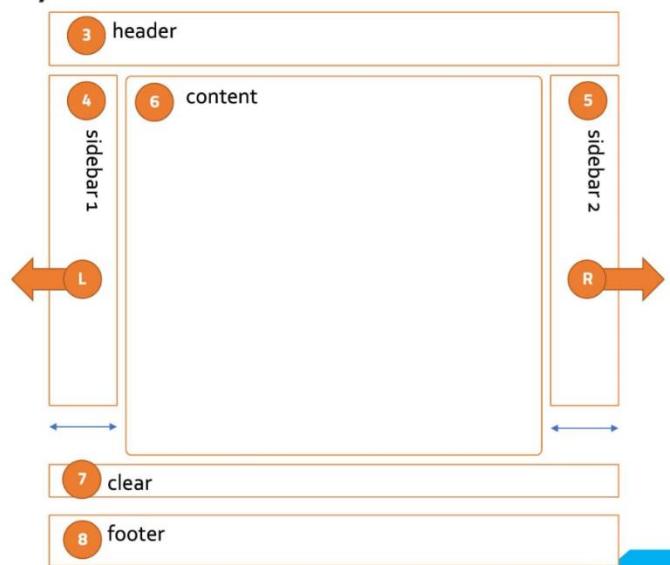
- due colonne, barra di navigazione a sinistra;
- stessa cosa ma destra;
- tre colonne.

Il layout fluido ridimensiona la pagina a seconda della larghezza della finestra del browser mentre in quello fisso rimane costante.

## Tipica struttura pagina desktop

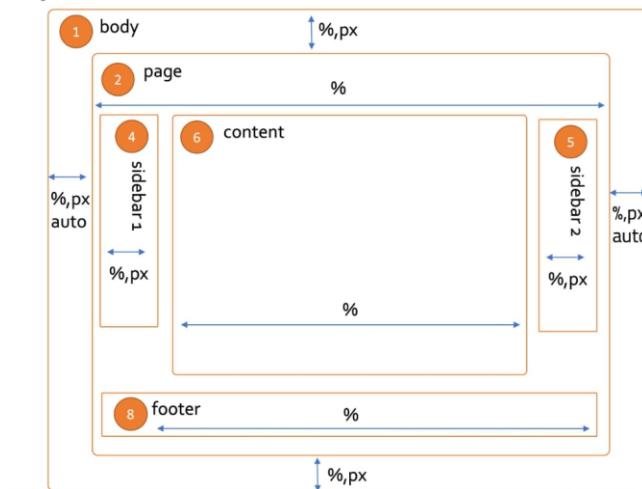


## CSS Layout a tre colonne

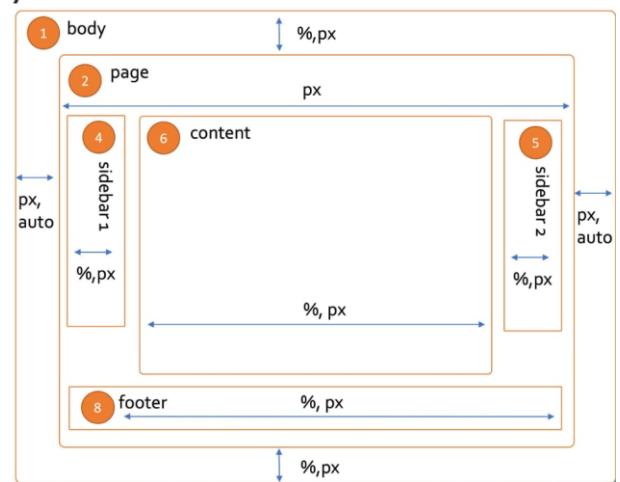


Il div **clear** impedisce al footer di sovrapporsi alle due sidebar se l'altezza del content è inferiore a quella delle sidebar. Per ottenere un layout a due colonne è sufficiente eliminare una delle sidebar.

### Layout fluido



### Layout fisso



### CSS 2:

Permette di specificare un file di stile diverso per tipologia di dispositivo, grazie all'attributo *media* al quale si può associare:

- **all** qualsiasi dispositivo
- **braille** dispositivi tattili
- **handheld** dispositivi mobili con risoluzione bassa
- **print** stampa
- **projection** proiettori
- **screen** schermi di computer
- **speech** sintetizzatori vocali
- **tty** dispositivi a caratteri
- **tv** televisori

## CSS 3:

È possibile specificare, oltre al tipo di dispositivo, le dimensioni min e max dello schermo per determinate regole. In questo modo si possono definire regole di base e regole che dipendono dalla dimensione dello schermo. Solitamente si usano le risoluzioni classiche (320, 480, 1024 px ecc).

## Gestire il layout:

Le regole del CSS possono agire su vari aspetti come il numero e la posizione del layout, la visibilità di alcune parti della pagina (come i menù) o la dimensione del font.

Solitamente si suddivide la pagina in colonne, normalmente 12, che vengono utilizzate per dividere la pagina in zone distinte.

Per comodità si fa in modo che border, margin e padding siano inclusi nella larghezza del contenuto tramite la regola:

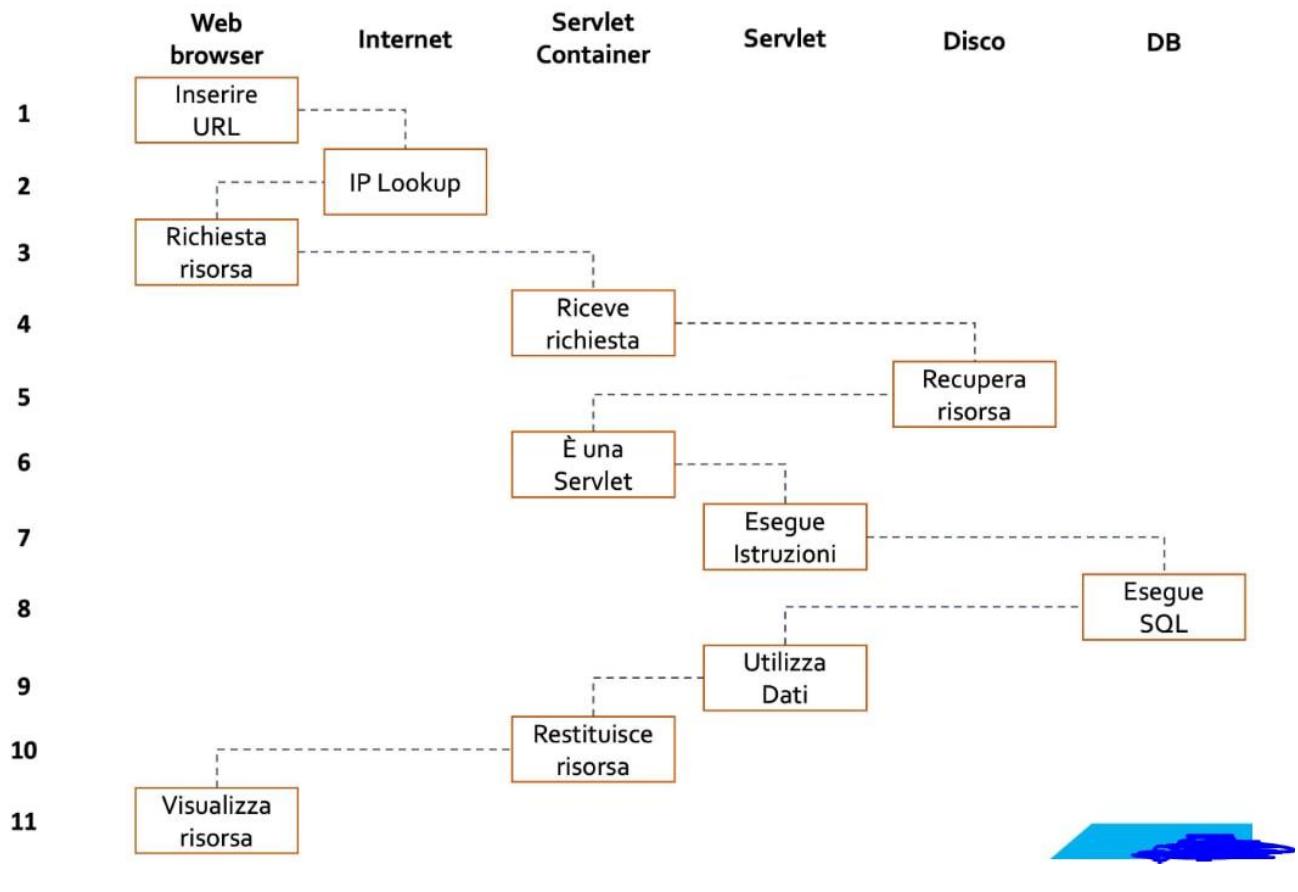
```
* {  
    box-sizing: border-box;  
}
```

Si crea una classe per la larghezza del contenuto di ciascun blocco misurata in colonne, in questo modo:

```
/* addossiamo automaticamente tutte le colonne a sx */  
[class*="col-"] {  
    float: left;  
}  
  
/* dimensioni delle colonne */  
.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}
```

Questo codice viene spesso inglobato nelle media queries.

# SERVER

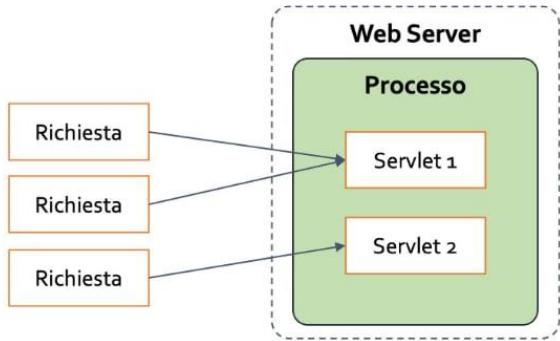


Il processo di creazione dell'HTML da parte del server è completamente invisibile al client. Una volta che la pagina viene restituita al client, il server ha terminato la sua esecuzione. Per comunicare eventuali cambi di stato nella pagina (es. input dell'utente) è necessario inviare una nuova richiesta HTTP.

## Servlet

Sono delle classi Java che permettono di gestire le richieste HTTP dinamicamente. Il codice HTML non è del tutto fisso ma buona parte viene generata in base a dei calcoli.

Il servlet container è un modulo all'interno di un web server che gestisce la vita delle servlet, tutto grazie ad un unico processo diviso in più threads gestiti in contemporanea.



All'interno di un'applicazione web sono presenti:

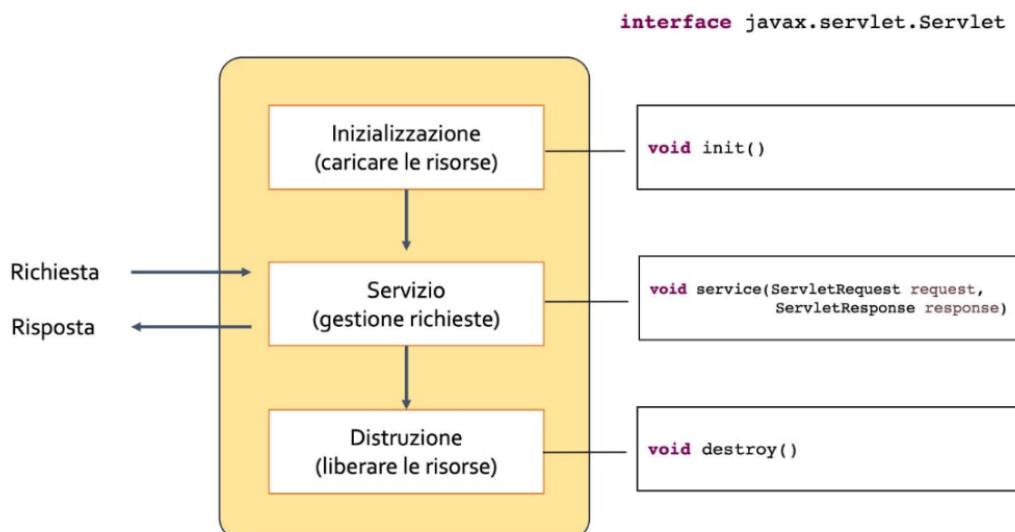
- file di configurazione (.xml): che permettono di configurarne le risorse;
- file statici: immagini, CSS, video, ecc;
- Servlet e JSP: permettono di creare contenuti dinamicamente;
- Classi e packages: normali file Java.

Si possono creare dei pacchetti autoinstallanti per il Servlet Container, chiamati WAR (Web Application Archive).

Le servlet offrono un supporto generale per architetture client-server.

Tutte le Servlet che gestiscono HTTP estendono la classe `javax.servlet.http.HttpServlet` facendone l'override dei metodi.

Ciclo di vita di una Servlet:



Di solito la init viene eseguita una sola volta. Le servlet create vengono tenute in memoria e in caso di necessità vengono distrutte per reinizializzarle all'accesso successivo.

```
class MyServlet  
extends HttpServlet  
  
    void doGet(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doPost(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doHead(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doPut(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doDelete(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doTrace(  
        HttpServletRequest request,  
        HttpServletResponse response)  
  
    void doOptions(  
        HttpServletRequest request,  
        HttpServletResponse response)
```

Una servlet non è un'applicazione Java indipendente ma è una componente di un'applicazione web e non è posizionata nella root del progetto ma nella cartella WEB-INF/classes.

La servlet deve essere dichiarata tramite i parametri:

- *servlet*: indica il nome;
- *servlet-mapping*: indica la corrispondenza tra la servlet e l'URL.

Queste informazioni possono essere contenute nel file *web.xml* o sotto l'annotazione *@WebServlet*.



La classe **HttpServletResponse** descrive le informazioni da restituire al client. I metodi *doGet()* e *doPost()* sono parametrici rispetto ad un'istanza di questa classe. Grazie a questo oggetto si può generare dinamicamente il contenuto con:

- *getWriter()*: restituisce un oggetto di tipo *PrintWriter* utile per scrivere contenuti testuali dinamici;
- *getOutputStream()*: restituisce un'istanza di *OutputStream* utile per scrivere contenuto dinamico di tipo binario (come immagini e video).

```

response.setContentType("text/html;charset=UTF-8");
try (PrintWriter out = response.getWriter()) {
    // generazione del codice della pagina
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet Test</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1> Test </h1>");
    out.println("</body>");
    out.println("</html>");
}
// il metodo "close" viene invocato in automatico
all'uscita della try

```

*HttpServletResponse* ha due metodi principali per aggiungere gli header della risposta HTTP:

- *addHeader*: aggiunge l'header con il valore specificato;
- *setHeader*: imposta il valore di un header già esistente con il valore specificato.

La gestione dello stato della risposta avviene tramite:

- *setStatus*: imposta lo stato di gestione della risposta (default: 200); se si verifica un'eccezione durante l'esecuzione della servlet viene impostato a 500;
- *sendRedirect*: invia un reindirizzamento a una nuova URL e imposta gli header necessari.

*HttpServletRequest* è una classe che descrive i dati inviati da un client (headers, parametri, altri dati).

Ha diversi metodi:

- *getHeaderNames*: scarica la lista di tutti gli header impostati dal client;
- *getHeaders*: restituisce i valori dell'header che ha il nome passato come parametro;
- *getParameter*: restituisce il valore del parametro specificato;
- *getParameterValues*: restituisce tutti i valori del parametro specificato;
- *getParameterMap*: restituisce una map con tutti i parametri;
- *getParameterNames*: restituisce un'enumerazione con tutti i nomi dei parametri impostati dal client.

Username	<input type="text"/>	Password	<input type="password"/>	<input type="button" value="Login"/>
----------	----------------------	----------	--------------------------	--------------------------------------

▪ HTML:

```
<form action="Test" method="post">
    <label for="username">Username</label>
    <input type="text" id="username" name="username" />
    <label for="username">Password</label>
    <input type="password" id="password" name="password" />
    <button type="submit">Login</button>
</form>
```

▪ Servlet:

```
String username = request.getParameter("username");
String password = request.getParameter("password");

if(username != null && password != null){
    // utilizzo dei valori
    this.login(username, password);
}
```

▪ HTML:

<input type="checkbox"/>	C	<input type="checkbox"/>	C++	<input type="checkbox"/>	Java	<input type="button" value="Invia"/>
--------------------------	---	--------------------------	-----	--------------------------	------	--------------------------------------

```
<form action="Test" method="get">
    <input type="checkbox" name="lang" value="c" /> C
    <input type="checkbox" name="lang" value="cpp" /> C++
    <input type="checkbox" name="lang" value="java" /> Java
    <button type="submit">Invia</button>
</form>
```

▪ Servlet:

```
String[] languages = request.getParameterValues("lang");
for (String lang : languages){
    out.println("<li>" + lang + "</li>");
}

// oppure

for (int i = 0; i < languages.length; i++){

    String lang= languages[i];
    out.println("<li>" + lang + "</li>");
}
```

## Validazione lato server:

Dato che l'utente non scrive sempre valori giusti è necessario implementare dei controlli che lo avvisino dell'errore. Solitamente si utilizzano le espressioni regolari.

## JSP

Quando il codice fisso HTML da generare è maggiore rispetto a quello variabile si utilizzano Java Server Pages. È possibile per una servlet importarle per generare l'output. Nella servlet si gestisce la logica applicativa e nella JSP la generazione output. Le JSP sono per la maggior parte pagine HTML contenenti tag speciali per la generazione di contenuti dinamici. Le pagine vengono sempre generate dal server e il browser riceve un HTML puro. Le JSP evitano la scrittura di codice ridondante e non serve definire il mapping dell'URL.

In passato si inseriva il codice Java all'interno delle JSP ma oggi si utilizzano apposite librerie: JSTL (insieme di tag XML) ed EL (metà linguaggio per rappresentare espressioni).

Per generare codice HTML dinamico si impostano alcune variabili e rendere il codice parametrico rispetto ai loro valori (cioè devi poter accedere a diverse variabili in modo dinamico). Con \${[nome variabile]} è possibile accedere alla variabile in modo dinamico (vedi pic sopra).

## Oggetti impliciti

- *pageContext*: l'intero contesto della JSP (come info sulla sessione, richiesta, risposta ecc);
- Oggetti ad “accesso rapido” (Map<String, String>): possono restituire i parametri della richiesta HTTP o i valori di un header specifico;
- Scoped variable: sono variabili di pagina, richiesta, sessione o applicazioni.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <ul>
      <c:forEach begin="0" end="5" var="i">
        <li>Elemento numero ${i}</li>
      </c:forEach>
    </ul>
  </body>
</html>
```

## Hello World!

- Elemento numero 0
- Elemento numero 1
- Elemento numero 2
- Elemento numero 3
- Elemento numero 4

■ Codice lato server

```
<ul>
  <c:forEach begin="0" end="5" var="i">
    <li>Elemento numero ${i}</li>
  </c:forEach>
</ul>
```

■ Output per il client

```
<ul>
  <li>Elemento numero 0</li>
  <li>Elemento numero 1</li>
  <li>Elemento numero 2</li>
  <li>Elemento numero 3</li>
  <li>Elemento numero 4</li>
</ul>
```

Elemento speciale, valutato lato server

Sostituito con il valore della variabile

È possibile suddividere il codice di una pagina su più file in questo modo:

```
<jsp:include page="header.jsp" />
```

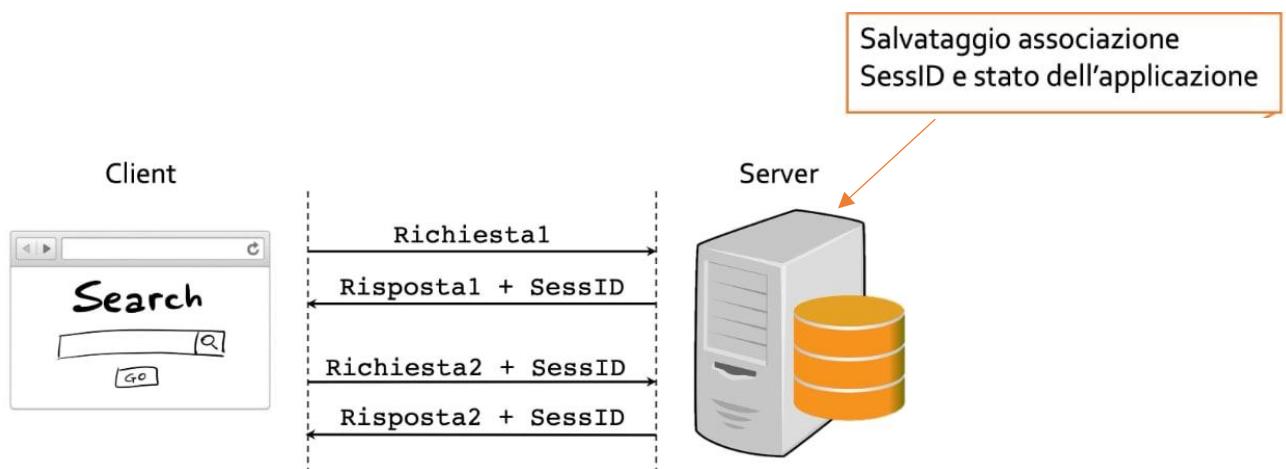
e importare delle classi per utilizzarle nelle JSP così:

```
<%@page import="java.util.List"%>
```

## Sessioni HTTP

I server web mettono a disposizione le sessioni le quali sono implementati sopra il protocollo HTTP. Una sessione è una sequenza di scambi HTTP legati tra loro secondo una logica applicativa; solitamente ha un limite temporale al termine del quale il server la considera scaduta. È possibile anche porre fine anticipatamente ad una sessione da parte dell'utente (esempio: logout).

Nella sessione sono conservate tutte le informazioni che permettono di mantenere lo stato dell'applicazione (utente corrente, parametri di ricerca ecc).



Tre possibili modi per inviare il token di sessione:

- Hidden form fields;
- URL rewriting;
- Cookies.

Le servlet supportano gli ultimi due. Esse supportano le sessioni tramite la API *HttpSession* la quale è un'interfaccia ad alto livello che usa i cookies se disponibili, altrimenti l'URL rewriting.

Come viene gestita la sessione:

- Si utilizza un oggetto della classe `javax.servlet.http.Session`;
- Lo si ricerca tramite la richiesta HTTP e se non si trova lo si inizializza (esempio: nuovo login);
- A questo punto l'oggetto offre la possibilità di registrare informazioni nella sessione sotto forma di coppia <chiave, valore>;
- La sessione si può terminare invalidando l'oggetto (logout).

Si può ricercare una sessione con metodo `request.getSession(true o false)`, la si crea con `request.getSession` senza parametri e la si distrugge con `request.invalidate`.

Per gestire i valori si possono usare i metodi `setAttribute`, `removeAttribute` e `getAttributes`.

#### Esempio: login

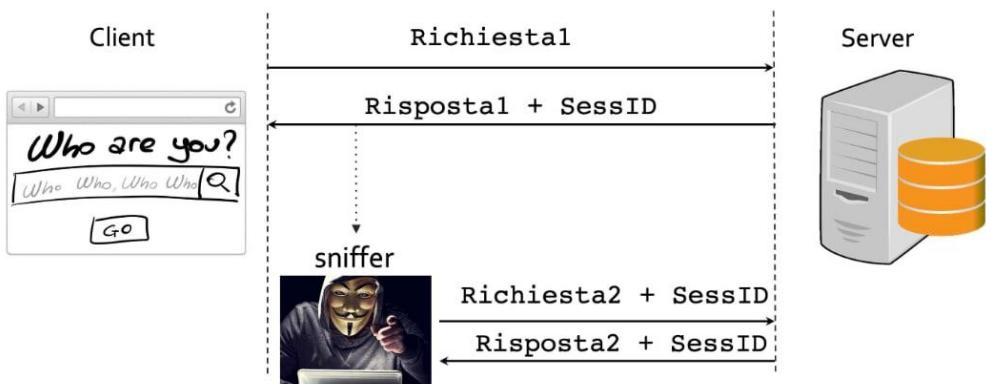
##### ■ Form HTML

```
<input type="text" id="username" name="username"/>
<input type="password" id="password" name="password"/>
```

##### ■ Servlet Login

```
HttpSession session = request.getSession();
String username = request.getParameter("username");
String password = request.getParameter("password");
if (username != null && password != null &&
    this.login(username, password)) {
    session.setAttribute("loggedIn", true);
}
■ Altre Servlet (controlla se l'utente è autenticato)
HttpSession session = request.getSession(false);
if(session.getAttribute("loggedIn").equals(true)){
    // utente autenticato
}
```

Il protocollo HTTP non è molto sicuro in quanto chiunque può impersonare un utente se ottiene il valore dell'ID della sua sessione, per questo è nato il protocollo HTTPS utilizzato per dati trasmettere dati sensibili tramite una connessione criptata.



## Scope

È possibile lavorare su tre livelli:

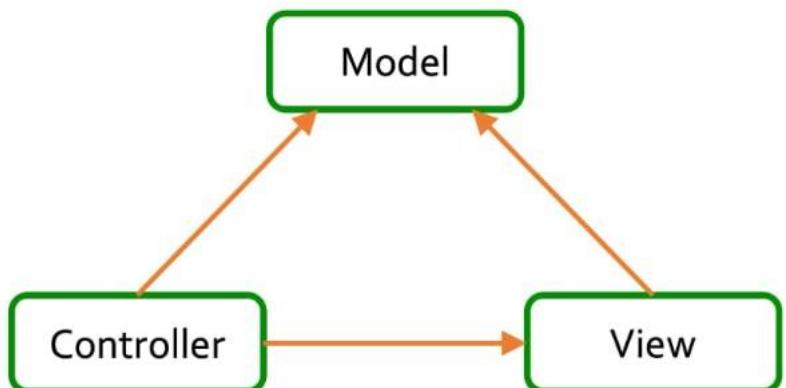
- Applicazione: informazioni globali condivise tra tutti gli utenti;
- Sessione: informazioni relative ad un singolo utente da condividere su più pagine;
- Pagina: informazioni necessarie solo per accedere alla stessa pagina.

## Model view controller

Nel web le interfacce cambiano spesso, ma la struttura dei dati no; per questo è meglio separare le due cose. Per farlo si utilizza un modello chiamato **model view controller**:

- *Model*: perché gestisce il comportamento e dati del dominio dell'applicazione;
- *View*: perché gestisce la visualizzazione dell'informazione;
- *Controller*: perché interpreta gli input dell'utente ricevuti attraverso la view e mostra l'eventuale cambio di stato all'utente sempre tramite sé stessa.

Queste sono le dipendenze tra gli elementi:

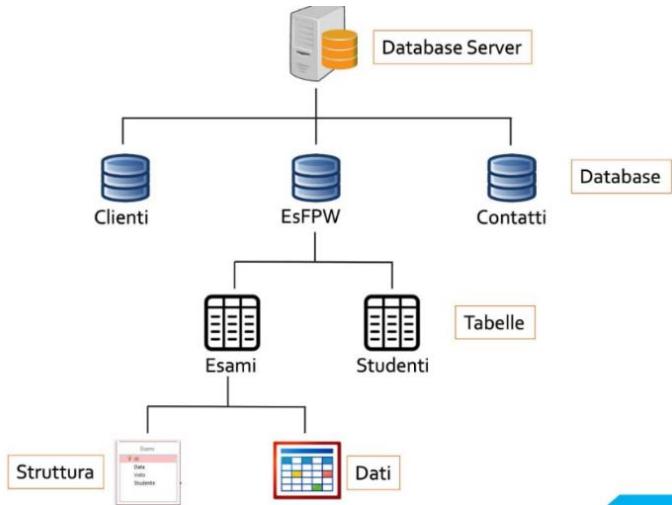


Questo modello può essere applicato grazie alle servlet insieme alle JSP.

## Database

Un database è una collezione strutturata di dati organizzati in maniera tale da poter essere ricercati efficacemente. I dati contenuti in un database vengono acceduti e modificati tramite il linguaggio standard SQL (Structured Query Language).

I dati di un database sono organizzati utilizzando delle tabelle. Un database server contiene un insieme di database i quali sono costituiti da un insieme di tabelle. Ogni tabella ha una sua struttura e contiene un insieme di dati messi in delle righe le quali contengono una serie di campi ognuno dei quali è associato ad una colonna.



Per creare delle tabelle in un database dobbiamo specificarne il nome della tabella insieme a un nome e a un tipo di dato per ogni colonna.

Non bisogna mai creare un unico tabellone per tutti i dati ma seguire delle regole per un buon design:

- Includere un identificatore univoco per ogni elemento della tabella;
- Ogni colonna deve contenere un singolo valore;
- Non ci devono essere colonne che ripetono lo stesso tipo di dato;
- Non ci devono essere dati ripetuti.

### Come creare una buona tabella:

1) **Identificare le entità** (ciò che va salvato nel database): tutto ciò che può essere riconosciuto da un identificatore unico è un'entità. Le entità possiedono degli attributi che le caratterizzano i quali possono avere diversi tipi. Una **chiave** è un campo di una tabella che identifica univocamente una riga, di solito si utilizza un campo apposito che si utilizza come chiave a cui si assegna un valore intero positivo. Una **chiave primaria** è una chiave con il numero minimo di campi.

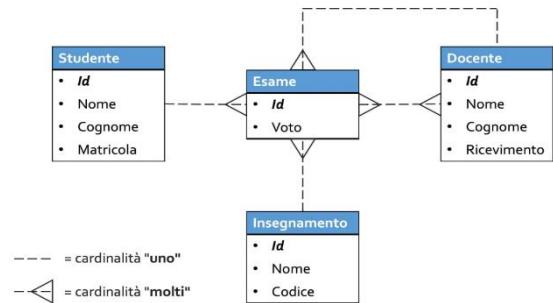
Studente
• <i>Id</i>
• Nome
• Cognome
• Matricola

Docente
• <i>Id</i>
• Nome
• Cognome
• Ricevimento

Insegnamento
• <i>Id</i>
• Nome
• Codice

Esame
• <i>Id</i>
• Voto

2) **Identificare le relazioni:** quando un'entità è collegata ad un'altra si dice che tra loro esiste una **relazione**, queste possono essere di tre tipi: uno a uno, uno a molti, molti a molti in base alla loro cardinalità. Una volta stabilita la cardinalità della relazione da A a B si stabilisce quella inversa da B ad A. In questo caso può esistere anche la relazione molti a uno. È possibile specificare degli attributi anche per le relazioni (esempio: una data associata alla relazione "matrimonio").



— = cardinalità "uno"  
—> = cardinalità "molti"

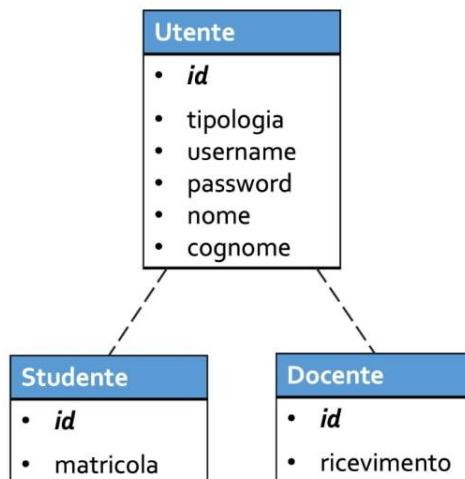
## Ereditarietà

È possibile modellare l'ereditarietà delle classi con le entità e le relazioni. Ci sono tre modi per farlo:

- 1) Entità unica per superclasse e sottoclasse:

Utente
• <i>id</i>
• tipologia
• username
• password
• nome
• cognome
• matricola
• ricevimento

- 2) Un'entità per la superclasse ed una per ogni sottoclasse:



- 3) Un'entità specifica per ognuna delle sottoclassi (i campi della superclasse sono duplicati su ogni tabella):

Studente
• <i>id</i>
• username
• password
• nome
• cognome
• matricola

Docente
• <i>id</i>
• username
• password
• nome
• cognome
• ricevimento

## Trasformare le entità in tabelle

Per ogni entità identificata si crea una tabella avente una colonna per attributo:

Docente			
id INT	nome VARCHAR(50)	cognome VARCHAR(50)	ricevimento VARCHAR(250)
...	...	...	...

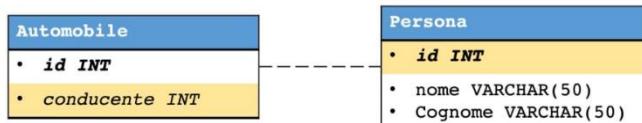
La rappresentazione usata finora rappresenta le colonne come un elenco.

Docente
• <b>id INT</b>
• <b>nome VARCHAR(50)</b>
• <b>cognome VARCHAR(50)</b>
• <b>ricevimento VARCHAR(250)</b>

Le relazioni si traducono in campi da inserire nelle tabelle (o in nuove tabelle)

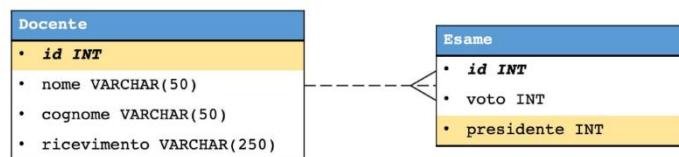
### Relazione uno ad uno

- Si crea una colonna nella tabella della prima entità che fa riferimento alla chiave primaria della seconda (vincolo di **chiave esterna**)



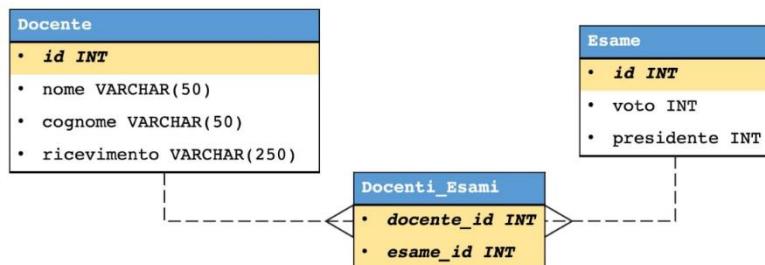
### Relazione uno a molti

- Si seleziona la tabella che rappresenta l'entità con cardinalità "molti"
- Si aggiunge una colonna che fa riferimento alla tabella dell'entità che ha cardinalità "uno" (vincolo di **chiave esterna**)



### Relazione molti a molti

- Si aggiunge una tabella apposita che contiene un riferimento alla prima tabella ed un riferimento alla seconda tabella



- Questa tabella non rappresenta un'entità, serve solo a modellare la relazione

## **Creazione e gestione di un database**

Il database server mantiene una lista di utenti che posso accedere ai vari database in esso contenuti. Esiste la possibilità di specificare diversi diritti di accesso (lettura, scrittura) su diversi database, per questo è necessario specificare le credenziali per connettersi. Ci sono tre modi per connettersi ad un database MySQL: command line, interfaccia grafica o API di linguaggi di programmazione. Una volta connessi si possono inviare dei comandi al database specificati tramite linguaggio standard SQL.

Per ognuno degli elementi del database è possibile eseguire quattro operazioni fondamentali:

- Create;
- Read;
- Update;
- Delete.

In gergo vengono chiamate CRUD.

### **SQL: creazione di un database**

Si usa il comando *CREATE DATABASE nomedatabase;*

Si crea un utente tramite il comando *CREATE USER 'nome'@'hostname' IDENTIFIED BY 'password'.*

Si possono assegnare diritti di lettura e scrittura ad un utente con *GRANT ALL ON esami.\* TO 'dino@localhost'.*

Per selezionare un database per le query successive si usa *USE nomedatabase.*

### **Tipi di dato**

I tipi di dato che si possono salvare all'interno delle tabelle sono:

- stringhe e testi
- numeri
- date
- formati binari
- Serial

Per i dati testuali si usano principalmente due tipi:

- CHAR: utilizza tutto lo spazio in memoria
- VARCHAR: ha dimensione variabile

Data type	Bytes used	Examples
CHAR( <i>n</i> )	Exactly <i>n</i> ( $\leq 255$ )	CHAR(5): "Hello" uses 5 bytes
		CHAR(57): "New York" uses 57 bytes
VARCHAR( <i>n</i> )	Up to <i>n</i> ( $\leq 65535$ )	VARCHAR(100): "Greetings" uses 9 bytes plus 1 byte overhead
		VARCHAR(7): "Morning" uses 7 bytes plus 1 byte overhead

Per i dati numerici:

Data type	Bytes used	Minimum value (signed/unsigned)	Maximum value (signed/unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
		-2147483648	2147483647
INT or INTEGER	4	0	4294967295
		-9223372036854775808	9223372036854775807
BIGINT	8	0	18446744073709551615
		-3.402823466E+38 (no unsigned)	3.402823466E+38 (no unsigned)
FLOAT	4	-1.7976931348623157E+308 (no unsigned)	1.7976931348623157E+308 (no unsigned)
		-1.7976931348623157E+308 (no unsigned)	1.7976931348623157E+308 (no unsigned)

- Per i booleani c'è **BOOL**, ma è un alias di **TINYINT**
- **SERIAL** è un alias di **BIGINT UNSIGNED** che viene utilizzato per gli ID univoci (chiavi)

Per le date:

Data type	Time/date format
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (Only years 0000 and 1901 - 2155)

Per i dati binari:

Data type	Bytes used	Attributes
TINYBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 255$ )	Treated as binary data—no character set
BLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 65535$ )	Treated as binary data—no character set
MEDIUMBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 16777215$ )	Treated as binary data—no character set
LONGBLOB( <i>n</i> )	Up to <i>n</i> ( $\leq 4294967295$ )	Treated as binary data—no character set

## Manipolazione delle tabelle

Le tabelle si possono creare tramite la sintassi SQL in questo modo:

- Creazione di una tabella:  
`CREATE TABLE studenti (  
 id SERIAL,  
 nome VARCHAR(128),  
 cognome VARCHAR(128),  
 matricola INT );`
- Si specifica prima il nome della tabella
- Poi si elencano, fra parentesi tonde, i nomi delle colonne con i rispettivi tipi di dato
- Utilizzare il tipo **SERIAL** imposta in modo automatico anche la chiave primaria della tabella
- Una tabella si cancella con la seguente sintassi  
`DROP TABLE studenti;`

Una volta creata la si può visualizzare con il comando DESCRIBE e modificarla col comando ALTER al quale si possono aggiungere diversi comandi:

- ADD: aggiunge una colonna;
- MODIFY: cambia il tipo di dato di una colonna;
- CHANGE: la rinomina o ne modifica il tipo di dato;
- DROP: la elimina;
- ADD PRIMARY KEY e ADD FOREIGN KEY specificano il vincolo di chiave primaria o esterna.

Il comando REFERENCES viene usato quando si dichiarano delle chiavi esterne.

Il comando ON UPDATE specifica cosa succede nel caso venga modificato il campo ID di una colonna puntata o cancellata la riga:

- CASCADE: la modifica viene effettuata anche nell'altra tabella in cascata;
- SET NULL: il campo dell'altra tabella viene messo a NULL;
- NO ACTION: non si fa nulla.

```
/* creiamo una tabella che abbia una chiave  
 * esterna verso la tabella studente  
 *  
 * Passo 1: creare la tabella  
 */  
CREATE TABLE esami (  
    id SERIAL,  
    voto INT,  
    studente_id BIGINT UNSIGNED );  
  
/*  
 * Passo 2: specificare il vincolo di chiave esterna  
 */  
ALTER TABLE esami ADD FOREIGN KEY studenti_fk (studente_id)  
REFERENCES studenti (id) ON UPDATE CASCADE;
```

Una volta creato l'insieme di tabelle bisogna popolarle con dei dati.

Per inserire una riga si usa il comando INSERT seguito da INTO nometabella.

Le colonne possono essere specificate in qualsiasi ordine.

L'attributo *default* è una parola chiave che inserisce il valore di default per il tipo selezionato.

## Ricerca di dati

Con il comando SELECT si seleziona la lista delle colonne, con il comando FROM il nome della tabella sul quale effettuare le ricerca e con WHERE i parametri della ricerca. Esempio:

```
SELECT id, nome, cognome, matricola  
FROM studenti  
WHERE nome = "Elisabetta" AND id = 4;
```

## Modifica di un dato

Funziona in questo modo: si ricerca una lista di righe e su quelle trovate si modifica il contenuto di una o più colonne.

```
UPDATE studenti SET  
matricola = 8373946, nome = "Gigi"  
WHERE id = 1 AND cognome = "Proietti";
```

- La tabella da modificare si specifica dopo la parola chiave UPDATE

## Cancellazione di dati

Circa come per l'update:

```
DELETE FROM studenti  
WHERE id = 9 AND cognome = "Locatelli";
```

## Ordinamento

Dopo aver filtrato gli elementi con SELECT \* FROM nometabella si usa il comando ORDER BY seguito dall'attributo sul quale si basa l'ordinamento desiderato. È possibile specificare più di un attributo.

## Contare elementi

Si utilizza il comando COUNT tra la select e la FROM e restituisce il numero di righe.

```
SELECT COUNT(*) FROM studenti  
WHERE matricola > 20000;
```

Al posto dell'asterisco di può specificare il nome di una colonna.

## Raggruppare righe

Il comando GROUP BY raggruppa più righe che hanno lo stesso contenuto su una colonna.

```
SELECT voto, COUNT(voto)  
FROM studenti GROUP BY voto;
```

- Restituisce una riga per ogni gruppo (ogni voto presente sulla tabella), con il numero di elementi del gruppo

voto	count(voto)
19	7
30	1



## Limitare il numero di righe

È possibile specificare dei limiti al numero di righe nella ricerca grazie al comando LIMIT. È possibile passare un solo numero, che verrà usato come limite superiore, oppure due numeri, uno come limite inferiore e l'alto che indica quante elencarne a partire dal primo.

## Ricerca con pattern

Nelle query SQL l'operatore LIKE nella WHERE permette di specificare solo una parte del valore tramite delle wildcards (caratteri che sostituiscono ad altri).

- Esempi
  - **LIKE 'Antoni\_'** seleziona sia Antonia che Antonio, ma non Antonietta
  - **LIKE 'Antoni%'** seleziona Antonia, Antonio e Antonietta
  - **LIKE '%Antoni%'** seleziona "Caro Antonio" e "Ciao Antonietta"
- Ricerchiamo tutti gli studenti il cui nome contenga una "i"
 

```
SELECT * FROM studenti
      WHERE nome LIKE '%i%';
```

## Ricerche e relazioni

Quando ci sono delle relazioni i dati sono suddivisi su più tabelle; per selezionarli si scrive **JOIN ... ON** il quale permette di unire virtualmente due tabelle per effettuare delle query sulla loro unione. Esempio:

```
SELECT studenti.cognome,
       studenti.nome,
       esami.voto
  FROM studenti JOIN esami
 WHERE esami.insegnamento_id = 19;
```

1

Studenti	<b>id</b>	<b>nome</b>	<b>cognome</b>	<b>matricola</b>
	1	Piero	Angela	123456
	2	Luciano	Onder	654321

2

**studenti JOIN esami ON studenti.id = esami.studente\_id**

<b>studenti.</b> <b>id</b>	<b>studenti.</b> <b>nome</b>	<b>studenti.</b> <b>cognome</b>	<b>studenti.</b> <b>matricola</b>	<b>esami.</b> <b>id</b>	<b>esami.</b> <b>studente_id</b>	<b>esami.</b> <b>insegnamento_id</b>	<b>esami.</b> <b>voto</b>
1	Piero	Angela	123456	1	1	19	24
2	Luciano	Onder	654321	2	2	19	27
1	Piero	Angela	123456	3	1	23	30
1	Piero	Angela	123456	4	1	25	18
2	Luciano	Onder	654321	5	2	23	30

3

**... WHERE esami.insegnamento\_id = 19;**

<b>studenti.</b> <b>id</b>	<b>studenti.</b> <b>nome</b>	<b>studenti.</b> <b>cognome</b>	<b>studenti.</b> <b>matricola</b>	<b>esami.</b> <b>Id</b>	<b>esami.</b> <b>studente_id</b>	<b>esami.</b> <b>insegnamento_id</b>	<b>esami.</b> <b>voto</b>
1	Piero	Angela	123456	1	1	19	24
2	Luciano	Onder	654321	2	2	19	27

4

**SELECT studenti.cognome, studenti.nome, esami.voto ...**

<b>studenti.</b> <b>nome</b>	<b>studenti.</b> <b>cognome</b>	<b>esami.</b> <b>voto</b>
Piero	Angela	24
Luciano	Onder	27

È possibile fare la JOIN con più di una tabella.

```
SELECT insegnamento.nome,  
       studente.cognome,  
       studente.nome,  
       esami.voto  
  
FROM esami JOIN insegnamento  
ON esami.insegnamento_id = insegnamento.id  
JOIN studenti  
ON studenti.id = esami.studente_id
```

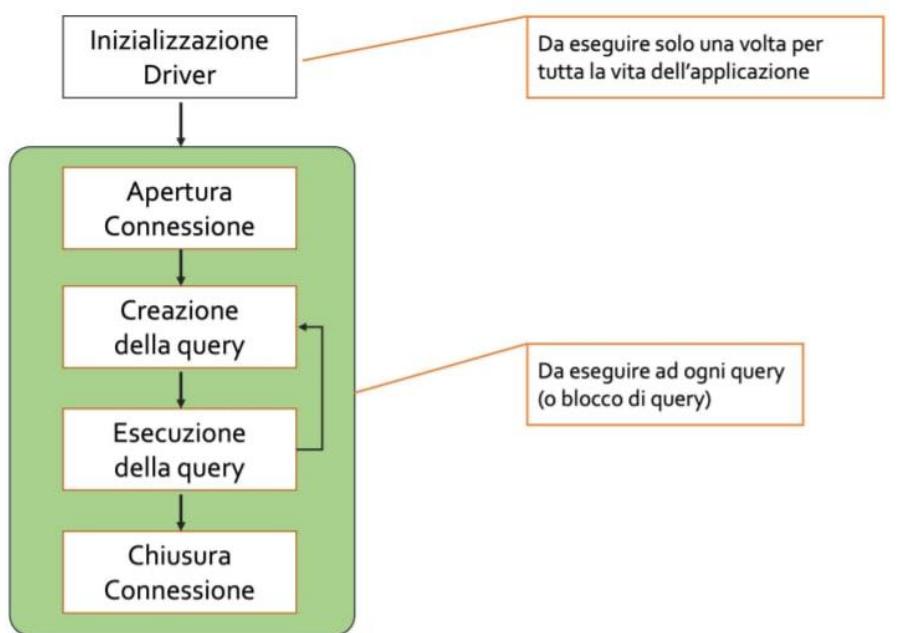
## Accesso al database tramite Java

### Java-Database Connectivity (JDBC)

è un API orientata ad oggetti per la connessione database indipendente dal database management system (DBMS) che permette la connessione, l'invio di query e la lettura dei risultati.

Ogni database ha un driver specifico caricato a runtime. Il package della libreria standard Java è java.sql.

### Schema di utilizzo JDBC



### Database Driver

Il jar delle classi del driver deve essere contenuto nel classpath dell'applicazione. Si aggiunge una libreria al progetto.

## Utilizzo del DB

```
try {
    // stringa di connessione formato
    String db = "jdbc:mysql://somehost:5432/somedb";

    // creazione e apertura della connessione
    // si specifica la URL, lo username e la password per il DB
    Connection conn = DriverManager.getConnection(db, "harry", "dobby");

    // utilizzo della connessione per inviare un comando sql
    Statement stmt = conn.createStatement();
    String sql = "SELECT * FROM esami";

    ResultSet set = stmt.executeQuery(sql);
    while (set.next()) {
        int id = set.getInt("id");
        System.out.println(id);
    }
    stmt.close();
    // chiusura della connessione
    conn.close();

} catch (SQLException ex) {
    // nel caso in cui la query fallisca (es: errori di sintassi)
    // viene sollevata una SQLException
    Logger.getLogger(Test.class.getName()).log(Level.SEVERE, null, ex);
}
```

NB: il numero di porta cambia in base al DBMS che stiamo usando

### Creazione della connessione

Si utilizza il metodo `getConnection()` che restituisce un oggetto di tipo `Connection`. Richiede tre parametri: `username`, `password` e la stringa di connessione (il quale formato dipende dal DBMS).

L'oggetto di classe `Connection` rappresenta la connessione al database e si utilizza per la connessione di comandi SQL; una volta che essi sono stati eseguiti la connessione deve essere chiusa con il metodo `close()`.

### Gestione degli errori

In caso di connessione fallita viene sollevata una SQL exception. I campi di eccezione ci forniscono informazioni sul problema:

- SQL state: id dell'errore;
- vendor code: codice specifico per il DBMS utilizzato.

### Esecuzione dei comandi SQL

1. Creare il comando SQL tramite una variabile stringa;
2. Creare una variabile di tipo `Statement` invocando il metodo `createStatement()` dell'oggetto `connection`;

3. A seconda del tipo di comando si invocano due metodi diversi:
  1. **SELECT**: metodo `executeQuery(String query)` restituisce un oggetto della classe `ResultSet`
  2. **INSERT, UPDATE, DELETE**: metodo `executeUpdate(String query)` restituisce il **numero** di righe modificate

4. Si invoca `close` sull'oggetto `statement`.

- Entrambi i metodi per eseguire i comandi SQL possono sollevare due tipi di eccezioni
  - **SQLException**: nel caso di errore di accesso al database, oppure se lo `statement` era stato chiuso in precedenza
  - **SQLTimeoutException**: se il database non risponde entro un determinato tempo. È una sottoclasse di `SQLException`

### Esecuzione di una insert:

```
try {
    Connection conn = DriverManager.getConnection(dbpath, "harry", "dobby");

    // creo lo statement
    Statement stmt = conn.createStatement();
    // definisco il comando sql
    String sql = "INSERT INTO studenti (id, nome, cognome, matricola) "
        + "VALUES (default, 'Emilio', 'Lussu', '2793342';

    // la eseguo: in rows ricevo il numero di righe inserite
    int rows = stmt.executeUpdate(sql);
    if(rows == 1){
        System.out.println("Insert ok!");
    }

    // chiudo lo statement
    stmt.close();

    // chiudo la connessione
    conn.close();

} catch (SQLException ex) {
    // nel caso in cui la insert fallisca (es: errori di sintassi)
    // viene sollevata una SQLException
    Logger.getLogger(Test.class.getName()).log(Level.SEVERE, null, ex);
}
```

### Esecuzione di una select (esempio incompleto):

```
// creo lo statement
Statement stmt = conn.createStatement();
// definisco la query sql
String sql = "SELECT * FROM studenti";

// la eseguo
ResultSet set = stmt.executeQuery(sql);

// ciclo sulle righe restituite
while (set.next()) {
    int id = set.getInt("id");
    String nome = set.getString("nome");
    String cognome = set.getString("cognome");
}

// chiudo lo statement
stmt.close();
// chiudo la connessione
conn.close();
```

## Result Set

È la classe che rappresenta l'insieme di risultati di una query. Ha un insieme di metodi `get[Tipo](String col)` che restituiscono il valore della colonna col del tipo specificato.

## Query con parametri

Per risolvere il problema del fatto che il contenuto e le tabelle di un database possono variare nel tempo, si usa la concatenazione di stringhe:

```
"SELECT id, matricola, nome, cognome FROM studenti  
WHERE matricola = " + myMatricola
```

## SQL Injection

L'SQL Injection è una tecnica di iniezione del codice usata per attaccare i database tramite il linguaggio SQL sfruttando il mancato controllo dell'input utente. Un esempio è inviare come password una stringa che confonda il database, il quale come effettua la ricerca risponde con esito positivo e permette di accedere senza la giusta password.

Per contrastarla si usano i Prepared statements, i quali possono venire creati dall'oggetto Connection tramite il metodo `prepareStatement()`:

- sono un oggetto apposito per rappresentare un comando SQL
- le parti parametriche vengono specificate esplicitamente
- la sintassi viene validata **prima** di eseguire la query con i valori veri dei parametri
- dunque, se viene inserito altro codice SQL, non va a buon fine la validazione
- servono principalmente per velocizzare comandi SQL da ripetere

I parametri si specificano con il metodo `set[Tipo](int pos, Tipo val)`.

## Transazioni

A volte è necessario eseguire alcune modifiche sul database per implementare una certa funzionalità. Esempio:



Invece di compensare manualmente le modifiche precedenti il database offre delle “transazioni”.

Si marcano con una sequenza di istruzioni SQL utilizzando due comandi:

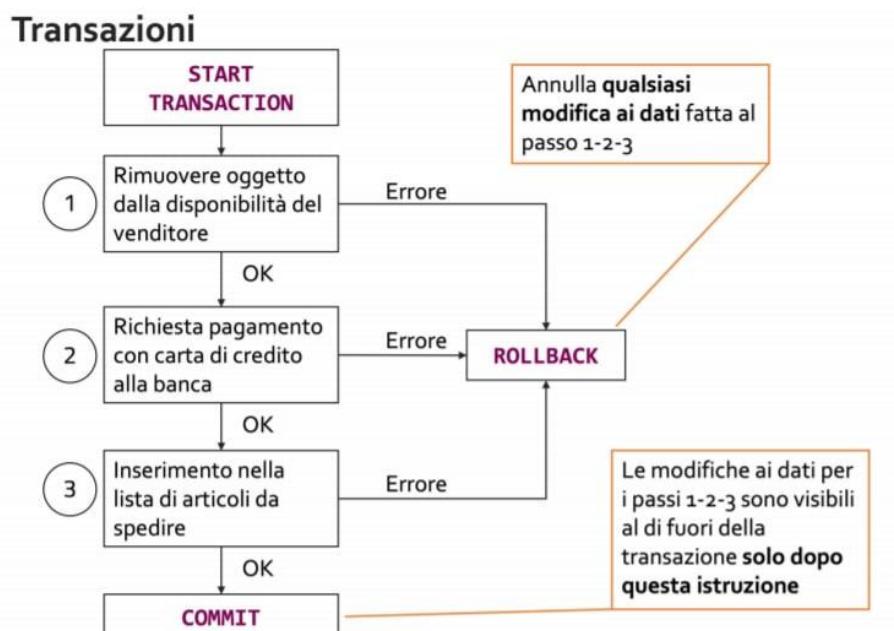
**START TRANSACTION**

...

**COMMIT**

La prima inizia una serie di istruzioni SQL che va considerata come atomica (le modifiche sono visibili fuori solo quando finisce l'esecuzione di tutto il codice), la seconda rende definitive tutte le modifiche.

Nel caso qualcosa vada storto il comando **ROLLBACK** permette di riportare il database allo stato precedente:

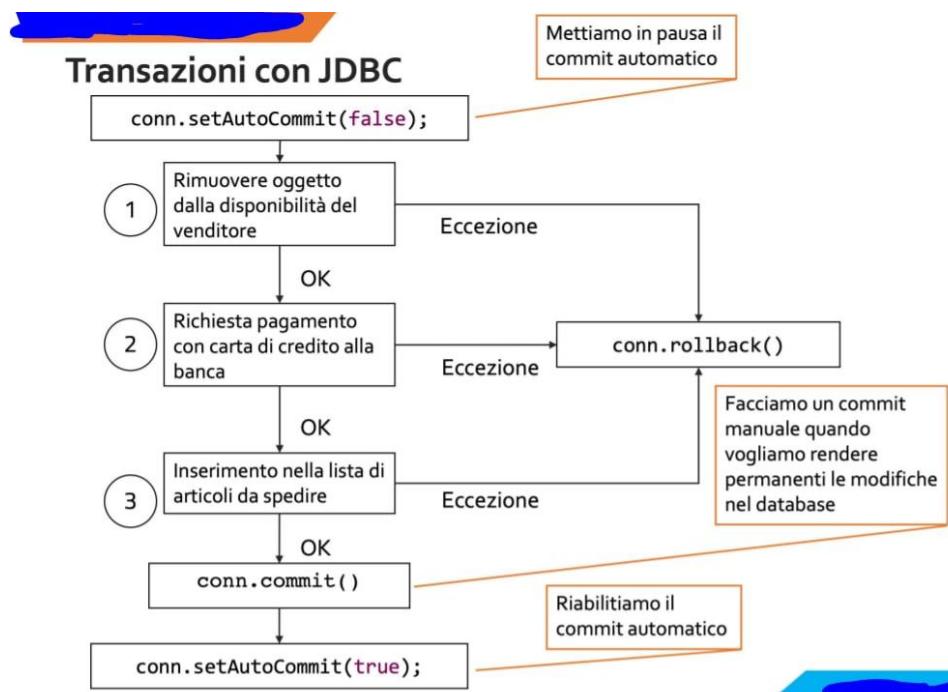


## Transazioni in java

Di norma tutti i comandi SQL che vengono inviati al database sono resi automaticamente permanenti, a meno che non si invochi il metodo `setAutoCommit(false)` sull'oggetto Connection.

Senza il commit automatico si può implementare una transazione così:

- il metodo `commit()` di `Connection` rende permanenti i comandi SQL
- il metodo `rollback()` di `Connection` annulla le modifiche dei comandi SQL eseguiti



## JavaScript

È un linguaggio di scripting, viene utilizzato all'interno di un programma per definire comportamenti particolari.

È un linguaggio interpretato, ogni browser esegue un interprete al suo interno ed espone al linguaggio le proprie caratteristiche tramite funzioni di libreria.

Ha una sintassi orientata agli oggetti ma non è un linguaggio Object Oriented in senso stretto.

Il codice Javascript di una pagina HTML può essere aggiunto in due modi:

- Includendo il sorgente direttamente nel codice della pagina HTML, tramite il tag **script**

- ```
<script type="text/javascript">
...
</script>
```

- Collegando la pagina HTML a file esterni, sempre con il tag **script**

- ```
<script src="myScript.js"></script>
```

La sintassi dei commenti è come quella del C.

Non ha bisogno di punti e virgola alla fine di ogni istruzione, ma solo se si ha una sola istruzione per riga.

Il primo carattere dei nomi delle variabili deve essere per forza una lettera, oppure i simboli \$ o \_ (niente numeri).

Una variabile può includere solo lettere a-z, A-Z, numeri 0-9 e \$ o \_.

Le stringhe si definiscono includendole in apici doppi o singoli, i quali si possono inserire dentro la stringa con il carattere \. Per concatenare più stringhe si usa il simbolo +.

I valori numerici sono assegnati alle variabili come in Java, ma le moltiplicazioni e le divisioni funzionano diversamente rispetto agli altri linguaggi (per esempio una divisione tra due interi restituisce un numero in virgola mobile e non un intero).

È possibile trasformare una variabile di un tipo in un altro tramite queste funzioni:

Change to type	Function to use	
Int, integer	parseInt()	n = 3.1415927
Bool, Boolean	Boolean()	i = parseInt(n)
Float, double, real	parseFloat()	document.write(i)
String	String()	
Array	split()	

Stampa 3,  
senza decimali

## Dynamic Typing

In Javascript è possibile che una variabile assuma tipi diversi a seconda del flusso di esecuzione. Una variabile non inizializzata ha un valore speciale: undefined.

```
// qui a e' un intero
a = 27;

if (b > 0) {
  // qui a e' una stringa
  a = "Poseidone"
}
```

## Espressioni

Operator(s)	Type(s)
() [] .	Parentheses, call, and member
++ --	Increment/decrement
+ - ~ !	Unary, bitwise, and logical
* / %	Arithmetic
+ -	Arithmetic and string
<< >> >>>	Bitwise
< > <= >=	Comparison
== != === !==	Comparison
& ^	Bitwise
&&	Logical
	Logical
? :	Ternary
= += -= *= /= %= <<= >>= >>>= &= ^=  =	Assignment
,	Sequential evaluation

A parte le cose già conosciute, ci sono delle differenze dai linguaggi più famosi:

- === controlla che due variabili siano uguali sia nel valore che nel tipo (mentre == converte allo stesso tipo prima di controllare l'uguaglianza);
- La guardia di uno switch può essere una stringa;
- Nei cicli non serve dichiara il tipo della variabile.

## Funzioni

Il passaggio dei parametri è fatto per valore, e l'interprete non si lamenta se non gli si passa il numero giusto di parametri.

Le funzioni hanno le stesse proprietà di quelle in Java.

Lo scope avviene come in C (e in Java), o sono locali o globali (quelle locali si dichiarano con la parola “var”, mentre quelle globali non la usano).

```
a = 5 // a e' una variabile globale
var b = 2 // b e' una variabile globale
if (a == 5)
    var c = 7 // c e' una variabile globale
```

```
function pluto()
{
    a = 8 // a e' una variabile globale
    var b = 6 // b e' una variabile locale
    if (a == 8)
        var c = 9 // c e' una variabile locale
}
```

Gli array funzionano in modo un po' diverso da Java:

- Si possono indicizzare per posizione;
- Possono essere associativi;
- Implementano diverse strutture dati (liste, code, pile ecc.).

Il resto funziona nello stesso modo (se passati per parametro si passa un riferimento, le matrici si formano con array di array ecc.).

```
// creazione di un array indicizzato per posizione
articoli = Array("il", "lo", "la");

// aggiunta di un elemento (in coda) all'array
articoli.push("i");

// rimozione dell'elemento in posizione 2 (indichiamo che
// vogliamo rimuovere un solo elemento)
articoli.splice(2,1);

// ciclo for su un array
for(var i = 0; i < articoli.length; i++)
{
    document.write(articoli[i] + '<br>');
}
```

## Numero variabile di parametri

Questa funzione stampa la lista di argomenti che viene passata:

```
function stampaParametri()
{
    for (j = 0 ; j < stampaParametri.arguments.length ; j++)
        document.write(stampaParametri.arguments[j] + "<br>")
}

stampaParametri("Athos", "Porthos", "Aramis");
```

O.O.

Ogni funzione contiene un campo chiamato *arguments* che contiene la lista di parametri passata alla funzione, quindi è possibile sapere quanti e quali sono i parametri passati alla funzione anche se non esplicitamente dichiarati.

## Array associativi e oggetti

In Javascript esistono gli array associativi, chiamati oggetti.

Vi si accede con una chiave alfanumerica al posto dell'indice (implementati tramite *HashMap*).

Tutto ciò che non è un booleano, un numero, una stringa o *undefined* è considerato oggetto.

Non esiste il concetto di classe come in Java, tuttavia c'è un concetto di classe che raggruppa oggetti che "nascono" con la stessa struttura.

È possibile numerare tutte le chiavi di un array associativo tramite il costrutto `for ... in`:

```
var lang = {  
    "en" : "Inglese",  
    "fr" : "Francese",  
    "it" : "Italiano"  
};  
lang["es"] = "Spagnolo";  
  
for(var l in lang)  
{  
    document.writeln("Codice lingua " + l + ":"  
        + lang[l] + "<br>");  
}
```

Codice lingua en: Inglese  
Codice lingua fr: Francese  
Codice lingua it: Italiano  
Codice lingua es: Spagnolo

## Chiusure lessicali

In Javascript una funzione può essere definita all'interno di un'altra, in tal caso l'insieme delle variabili riferite della funzione e il loro stato viene chiuso (mantenuto vivo) insieme alla funzione stessa. Esempio:

```
function Counter(){  
    // questa variabile continuera' a esistere dopo l'uscita dallo scope  
    var count = 1;  
  
    function incrCounter(){  
        count = count +1;  
        return count;  
    }  
  
    return incrCounter;  
}  
  
// questa variabile contiene la funzione incrCounter  
var cnt = Counter();  
  
// chiamo per la prima volta l'incremento, la variabile count da 1  
// passa a 2. Da notare che e' stata definita nella funzione Counter  
// e non nella funzione incrCounter (stampa 2)  
document.writeln("count, prima chiamata: " + cnt() + "<br>");  
  
// il valore di count passa da 2 a 3 (stampa 3)  
document.writeln("count, seconda chiamata: " + cnt() + "<br>");
```

## Definizione di oggetti

Gli oggetti in Javascript possono essere definiti in due modi: fornendo una funzione che rappresentante il costruttore degli oggetti di uno stesso tipo o definendone la struttura “al volo” e assegnandola ad una variabile tramite la *short object notation* (questo metodo permette di creare solo un oggetto dello stesso tipo).

Essendo le funzioni dei valori non vi è distinzione tra attributi e metodi, ed è possibile cambiarle. Per renderle non modificabili bisogna assegnarli al campo *prototype* del costruttore, il quale è condiviso da tutti gli oggetti creati tramite quel costruttore.

Al momento della chiamata, se l'interprete non trova la funzione direttamente nell'oggetto, la ricerca nel prototype, il quale si richiama con il punto (.).

Variabili e metodi statici possono essere creati utilizzando il campo *constructor*.

## Document Object Model (DOM)

Il codice Javascript e l'HTML comunicano grazie al *Document Object Model*, il quale è una rappresentazione ad oggetti della struttura ad albero della pagina HTML.

Javascript ha un oggetto speciale, messo a disposizione da ogni browser, chiamato *document*.

Il *document* è la radice della rappresentazione ad albero della pagina HTML. Ha delle funzioni per creare dei nuovi nodi dinamicamente, e contiene delle funzioni per ricercare i nodi di base. Le altre funzioni sono comuni a tutti i nodi dell'albero.

### Node

Rappresenta un nodo del documento HTML, il quale può essere:

- Il documento stesso;
- Un elemento;
- Un attributo;
- Del testo;
- Un commento.

Ha delle funzioni per:

- Elencare, aggiungere o eliminare i figli;
- Accedere al nodo padre;
- Elencare gli attributi;
- Accedere/modificarne il valore.

Proprietà:

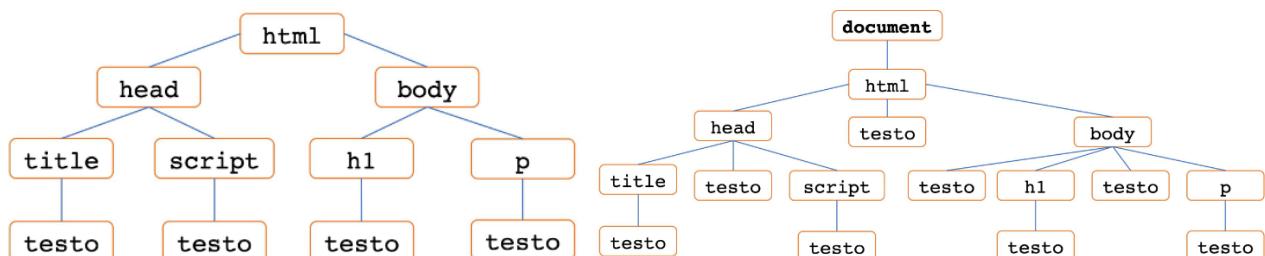
- innerHTML: restituisce il codice html contenuto all'interno del documento;
- innerText: restituisce il testo contenuto all'interno dell'elemento.

```

<!DOCTYPE html>
<html>
<head>
    <title>Proviamo insieme Javascript!</title>
    <script type="text/javascript">
        document.getElementById("titolo");
    </script>
</head>
<body>
    <h1 id="titolo">Un gran bel titolo</h1>
    <p id="paragrafo">Un gran bel paragrafo</p>
</body>
</html>

```

## Struttura ad albero (concettuale e reale):



## Modifica CSS

Si possono cambiare non solo i nodi, ma anche le proprietà del CSS, modificando i valori delle proprietà, aggiungere o rimuovere classi a diversi elementi ecc.

Ogni nodo di tipo elemento ha un oggetto *style*.

È possibile aggiungere o rimuovere delle classi ad un elemento tramite l'attributo `className`.

- Modifica del colore del testo del titolo

```
document.getElementById('titolo').style.color = 'red';
```

- Aggiunta di una classe CSS

```
document.getElementById('titolo').className += ' blueColor';
```

Un evento può essere un qualcosa come il click di un utente su un bottone o il passaggio del mouse su un elemento dinamico.

Questi eventi sono gestibili tramite Javascript.

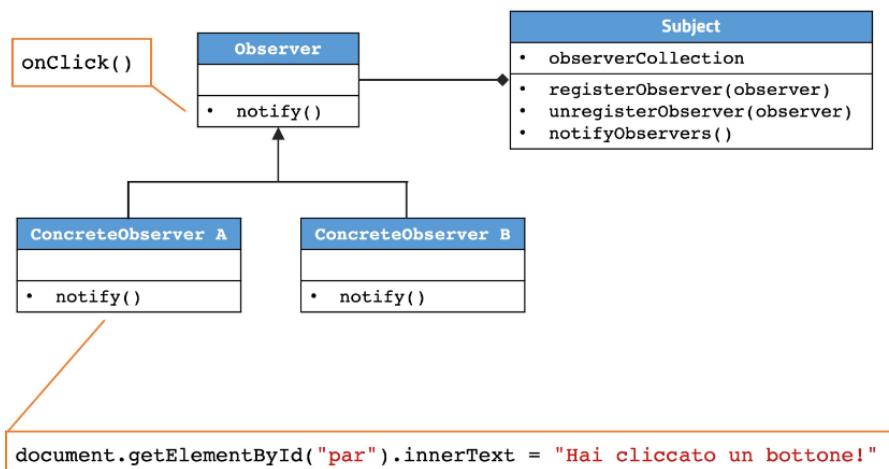
Mostra un messaggio

Hai cliccato un bottone!

- Codice Javascript da eseguire (**handler**)

```
document.getElementById("par").innerText =  
"Hai cliccato un bottone!"
```

- Quando? Al click del bottone (**evento**)



## Come agganciare gli handler

Si può associare un *handler* per un dato evento specificando una funzione Javascript che se ne occupi. Questa funzione si aggancia all'elemento HTML specificando il valore corrispondente di un attributo. Si può inoltre assegnare un handler tramite la corrispondente variabile Javascript.

Il browser chiama per noi la funzione quando l'evento si innesca.

Tutti gli eventi sono parametrici rispetto a un *event object* che contiene informazioni addizionali rispetto all'evento (posizione del puntatore, pulsante premuto ecc.).

```
function addClick(event){  
    //clickCount e' una variabile globale  
    clickCount++;  
    event.currentTarget.innerHTML = 'Click: ' + clickCount;  
}
```

- Aggiunta direttamente nell'HTML  
`<p id="paragrafo" onclick="addClick(event)">`
- Oppure aggiunta tramite Javascript  
`document.getElementById('paragrafo').onclick = addClick;`

I due metodi dell'esempio sono diversi semanticamente tra loro:

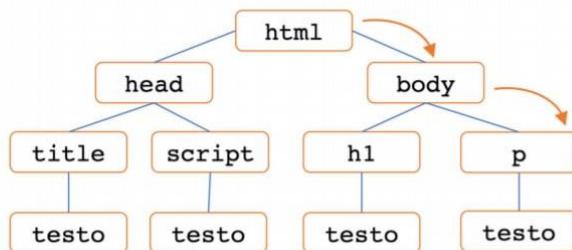
- Il primo modo (HTML) è equivalente a scrivere una funzione che contiene l'istruzione che invoca la funzione **addClick**. Questa funzione viene assegnata alla proprietà **onclick** del nostro elemento
- Il secondo assegna la funzione **addClick** ad un campo dell'elemento del DOM, che ne diventa il proprietario quando l'evento si scatena

Il *this* in Javascript rappresenta sempre il proprietario della funzione al momento della chiamata.

## Eventi

Event tunneling:

- Quando si clicca su un box, il supporto cerca l'elemento **più interno** che contiene il punto cliccato

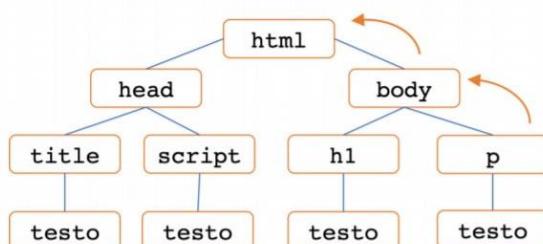


Questo è un paragrafo

Event bubbling:

### Event bubbling (click)

- Dopo di che l'evento "risale" la gerarchia cercando qualcuno che lo gestisca
- Di default viene richiamato ogni handler che si trova



Questo è un paragrafo

## Event target e current target:

- All'interno della variabile **event.target** viene mantenuto l'elemento più "profondo" nell'albero DOM che ha iniziato il bubbling dell'evento
- Invece in **event.currentTarget** viene mantenuto l'elemento che sta gestendo l'evento (cioè che ha associato l'handler in corso di esecuzione)
- Dunque **event.currentTarget** ed **event.target** possono contenere due valori differenti quando l'evento viene gestito tramite bubbling
- Es: nell'albero precedente **event.target** è sempre il **p**, mentre **event.currentTarget** dipende dall'handler
- Se **event.target** e **event.currentTarget** hanno lo stesso valore allora siamo nell'elemento più profondo della gerarchia

## Controllare l'event object:

*event.preventDefault()* fa modo che le azioni che un browser fa di default non vengano eseguite, mentre *event.stopPropagation()* interrompe la propagazione di un determinato evento ad altri elementi della pagina.

## Funzioni e proprietà di DOM

Ogni nodo ha anche determinate caratteristiche:

- proprietà;
- eventi;
- funzioni.

## Form

### Form Object Properties

Property	Description
<u>acceptCharset</u>	Sets or returns the value of the accept-charset attribute in a form
<u>action</u>	Sets or returns the value of the action attribute in a form
<u>enctype</u>	Sets or returns the value of the enctype attribute in a form
<u>length</u>	Returns the number of elements in a form
<u>method</u>	Sets or returns the value of the method attribute in a form
<u>name</u>	Sets or returns the value of the name attribute in a form
<u>target</u>	Sets or returns the value of the target attribute in a form

### Form Object Methods

Method	Description
<u>reset()</u>	Resets a form
<u>submit()</u>	Submits a form

## Form Events

Attribute	Description
<u>onblur</u>	The event occurs when a form element loses focus
<u>onchange</u>	The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)
<u>onfocus</u>	The event occurs when an element gets focus (for <label>, <input>, <select>, <textarea>, and <button>)
<u>onreset</u>	The event occurs when a form is reset
<u>onselect</u>	The event occurs when a user selects some text (for <input> and <textarea>)
<u>onsubmit</u>	The event occurs when a form is submitted

## Mouse Events

Property	Description
<u>onclick</u>	The event occurs when the user clicks on an element
<u>ondblclick</u>	The event occurs when the user double-clicks on an element
<u>onmousedown</u>	The event occurs when a user presses a mouse button over an element
<u>onmousemove</u>	The event occurs when the pointer is moving while it is over an element
<u>onmouseover</u>	The event occurs when the pointer is moved onto an element
<u>onmouseout</u>	The event occurs when a user moves the mouse pointer out of an element
<u>onmouseup</u>	The event occurs when a user releases a mouse button over an element

## Keyboard Events

Attribute	Description
<u>onkeydown</u>	The event occurs when the user is pressing a key
<u>onkeypress</u>	The event occurs when the user presses a key
<u>onkeyup</u>	The event occurs when the user releases a key

## Frame/Object Events

Attribute	Description
<u>onabort</u>	The event occurs when an image is stopped from loading before completely loaded (for <object>)
<u>onerror</u>	The event occurs when an image does not load properly (for <object>, <body> and <frameset>)
<u>onload</u>	The event occurs when a document, frameset, or <object> has been loaded
<u>onresize</u>	The event occurs when a document view is resized
<u>onscroll</u>	The event occurs when a document view is scrolled
<u>onunload</u>	The event occurs once a page has unloaded (for <body> and <frameset>)

## Proprietà Event Object

Property	Description
<u>altKey</u>	Returns whether or not the "ALT" key was pressed when an event was triggered
<u>button</u>	Returns which mouse button was clicked when an event was triggered
<u>clientX</u>	Returns the horizontal coordinate of the mouse pointer, relative to the current window, when an event was triggered
<u>clientY</u>	Returns the vertical coordinate of the mouse pointer, relative to the current window, when an event was triggered
<u>ctrlKey</u>	Returns whether or not the "CTRL" key was pressed when an event was triggered
<u>keyIdentifier</u>	Returns the identifier of a key
<u>keyLocation</u>	Returns the location of the key on the advice
<u>metaKey</u>	Returns whether or not the "meta" key was pressed when an event was triggered
<u>relatedTarget</u>	Returns the element related to the element that triggered the event
<u>screenX</u>	Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered
<u>screenY</u>	Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered
<u>shiftKey</u>	Returns whether or not the "SHIFT" key was pressed when an event was triggered

## Jquery

Il meccanismo di gestione della dinamicità delle pagine con il Javascript ha bisogno di una interfaccia ad alto livello per risolvere le differenze tra le API dei diversi browser. Per questo sta diventando uno standard jQuery.

- API pulita per la manipolazione del DOM
- Manipolazione dei CSS
- Gestione degli eventi HTML
- Animazioni ed effetti
- Tecniche AJAX (per la gestione degli aggiornamenti asincroni della pagina)
- Funzioni di utilità

Per implementarlo nel documento:

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

jQuery offre una soluzione comoda per selezionare gli elementi con una sintassi pulita e concisa.

## La funzione \$

I selettori jQuery si utilizzano tramite la funzione \$ e permettono di selezionare elementi HTML in base a id, classi CSS, attributi, valore degli attributi ecc.. Il risultato della selezione è una lista di elementi del DOM “aumentata” con alcune funzioni di utilità offerte da jQuery, che spesso permettono di manipolare l’intera lista in un colpo solo.

Es: nascondere tutti i paragrafi di un documento HTML

- `$( "p" ).hide();`
- una riga di codice !
  
- `$( "*" )`: seleziona tutti gli elementi
- `$( this )`: seleziona l’elemento HTML corrente
- `$( "p.intro" )`: seleziona tutti gli elementi `<p>` con `class="intro"`
- `$( "p:first" )`: seleziona il primo elemento `<p>`
- `$( "ul li:first" )`: seleziona il primo `<li>` dentro il primo `<ul>`
- `$( "ul li:first-child" )`: seleziona il primo `<li>` dentro ogni `<ul>`
- `$( "[href]" )`: seleziona tutti gli elementi con attributo `href`
- `$( "a[target='_blank']" )`: seleziona tutti gli elementi `<a>` con `target="_blank"`
- `$( "a[target!='_blank']" )`: seleziona tutti gli elementi `<a>` con target diverso da `_blank`
- `$( ":button" )`: seleziona tutti gli elementi `<button>` e gli elementi `<input>` con `type="button"`
- `$( "tr:even" )`: seleziona tutti gli elementi `<tr>` in posizione pari
- `$( "tr:odd" )`: seleziona tutti gli elementi `<tr>` in posizione dispari

## jQuery: manipolazione DOM

Una volta che abbiamo selezionato uno o più elementi si possono effettuare varie manipolazioni del DOM. Le funzioni per fare get e set di un valore sono le stesse di solito, se viene specificato un valore viene eseguita una set, altrimenti una get, funzioni importanti sono:

- `text()`: imposta o restituisce il testo contenuto nell’elemento/elementi selezionati
- `html()`: imposta o restituisce l’HTML (tutto il markup) per l’elemento/ elementi selezionati
- `val()`: imposta o restituisce il valore degli elementi di input (qualsiasi sia il loro tipo)
- `attr(name)`: imposta o restituisce il valore di un attributo (passato per parametro)

- Cambiamo il testo interno all'elemento con id **hd-news**  
`$("#hd-news").text("Io arrivo da jQuery");`
- Cambiamo il path di un'immagine con id **img-news** (N.B. questo ha l'effetto di sostituire l'immagine visualizzata)  
`$("#img-news").attr("src","img/newimg.jpg");`
- Salviamo il valore contenuto in un elemento input con id **id-news** all'interno di una variabile  
`var input_val = $("#id-news").value();`
- Sostituiamo la definizione dell'HTML interno ad un elemento con id **txt-news** passando direttamente codice HTML  
`$("#txt-news").html("<h2>Mega news!!</h2><p>Ho passato il parziale!</p>");`

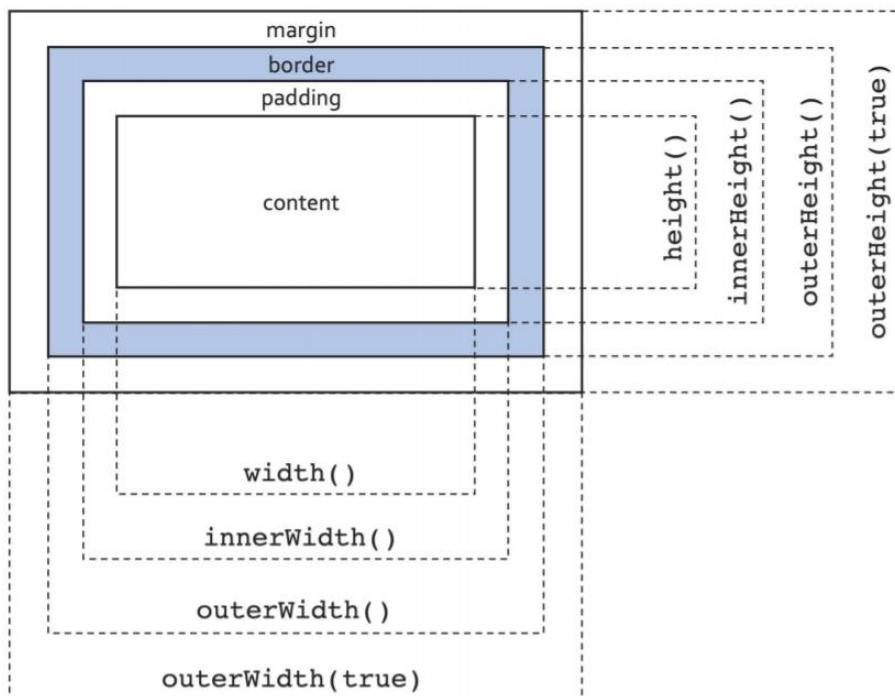
jQuery permette di inserire nuovi elementi tramite quattro funzioni che prendono per parametro diversi oggetti: testo semplice, HTML, oggetti jQuery, elementi DOM creati con codice JavaScript. Queste funzioni sono:

- **append()**: inserisce il contenuto **all'interno** di ognuno degli elementi selezionati, in **ultima** posizione
- **prepend()**: inserisce il contenuto **all'interno** di ognuno degli elementi selezionati, in **prima** posizione
- **after()**: inserisce il contenuto **dopo** ognuno degli elementi selezionati (come fratello)
- **before()**: inserisce il contenuto **prima** di ognuno degli elementi selezionati (come fratello)

Per quanto riguarda la rimozione degli elementi abbiamo le funzioni **empty()** e **remove()** la prima elimina tutti i figli dell'elemento selezionato dal DOM, la seconda elimina dal DOM l'elemento selezionato e tutti i suoi figli.

- `$("#ol").remove();`  
~~<ol>~~  
~~<li>List item 1</li>~~  
~~<li>List item 2</li>~~  
~~<li>List item 3</li>~~  
~~</ol>~~
- `$("#ol").empty();`  
~~<ol>~~  
~~<li>List item 1</li>~~  
~~<li>List item 2</li>~~  
~~<li>List item 3</li>~~  
~~</ol>~~

Per quanto riguarda le classi CSS jQuery ci mette a disposizione le funzioni **addClass()** e **removeClass()** per aggiungerle e rimuoverle, mentre per quanto riguarda la loro manipolazione abbiamo la funzione **css()** che ci permette di accedere e leggere il valore degli stili. Per quanto riguarda le dimensioni dei box abbiamo delle funzioni che li restituiscono, ovvero:



## Event Handlers

- Gli event handlers si agganciano sempre utilizzando delle funzioni particolari sugli elementi selezionati
- Un modo è utilizzare la funzione **on()**, specificando il nome dell'evento (senza "on" davanti) e la funzione da agganciare
 

```
$('#foo').on('click', function () {
    alert('Hai clickato su "foo."');
});
```
- Si possono anche specificare più eventi in un colpo solo (es: aggiungere o rimuovere la classe "entered" nell'elemento con id "foo")
 

```
$('#foo').on('mouseenter mouseleave', function () {
    $(this).toggleClass('entered');
});
```
- jQuery mette anche a disposizione delle funzioni alias per la **on()**, che si chiamano direttamente con il nome dell'evento, es: **click()**

```
$('#foo').click(function () {
    alert('Hai clickato su "foo."');
});
```

Le istruzioni per agganciare gli event handlers si effettuano solitamente al termine del caricamento del documento, in jQuery basta specificare una funzione per l'evento *ready* del *document*.

- In gran parte dei siti che utilizzano jQuery troverete questo schema:

```
$(document).ready(function () {
    // istruzioni per agganciare event handlers
    $("#next-news").click(function () {
        // ...
    });
    // ...

    // altre funzioni
    function changeNews(news) {
        // ...
    }
});
```

## AJAX (Asynchronous Javascript and XML)

Sono tecniche di sviluppo tramite Javascript che consentono di scambiare dati con il server senza ricaricare la pagina. Consentono una migliore interfaccia utente e in moltissimi casi riducono il traffico di rete. Nasce dall'introduzione dell'oggetto XMLHttpRequest nella API Javascript; questo oggetto consente di inviare delle richieste HTTP al server quando la pagina è stata già caricata direttamente da codice Javascript e in modo totalmente asincrono. Questo implica che si possono richiedere dati solo quando sono necessari, senza caricarli da subito, e che la pagina viene modificata in base a cosa fa l'utente senza ricaricarla interamente, così l'applicazione web diventa veramente dinamica (es. Google Maps).

### XmlHttpRequest

- Non è uniforme su tutti i browser
  - in alcune versioni di IE si crea in modo differente (strano...)
- Permette di ricevere dati
  - testuali
  - in formato XML
- Noi useremo dei dati testuali
  - ma codificati in modo furbo (Javascript Object Notation)
- Permette di inviare delle richieste **asincrone** al server
  - cioè l'esecuzione del codice non si blocca in attesa della risposta del server
  - si registra una funzione che l'interprete richiama non appena la risposta arriva
- Ha un'interfaccia per il programmatore un po' confusionaria
- Chi ci aiuta ancora una volta?
- Noi continuiamo a utilizzare una sintassi unificata offerta da jQuery

## JSON

- Il formato testuale che si utilizza di solito per scambiare informazioni con il server tramite Ajax si chiama **JSON**
- Che sta per **JavaScript Object Notation**

```
{  
    "id" : 1,  
    "heading": "Dossier Moderna oggi...",  
    "img": "http://ansa/speranza.jpg",  
    "text": "Il Ministro della Salute..."  
}
```

- L'abbiamo già incontrata, è la short object notation con qualche restrizione in più (non ci possiamo mettere funzioni)
- Ci permette di non dover decodificare i valori quando arrivano al Javascript
- La codifica degli array in JSON è supportata anche da Java, quindi non dobbiamo sforzarci troppo nemmeno lato server
- jQuery ci mette a disposizione una funzione molto comoda che ci permette di gestire le differenti implementazioni in modo trasparente
- La funzione prende per parametro un solo oggetto in JSON
- Al suo interno si possono specificare un insieme di campi che permettono di configurare la richiesta
- I parametri di base sono:
  - **url**: indirizzo al quale inviare la chiamata (default: pagina corrente)
  - **success**: la funzione da chiamare nel caso la richiesta vada a buon fine
  - **error**: la funzione da chiamare nel caso la richiesta fallisca
  - **data**: i dati da passare al server via POST o GET
  - **dataType**: la codifica dei dati inviati dal server (per noi sarà JSON)
  - **asynch(true/false)**: se impostato a **false** si attende la risposta del server prima di proseguire con altre istruzione Javascript dopo la chiamata Ajax, se a **true** tutta la gestione avviene in asincrono (default: true)

### La funzione `$.ajax()`

jQuery mette a disposizione una funzione che permette di gestire le implementazioni in modo trasparente, la funzione `ajax()`. Prende per parametro un solo oggetto in JSON. I parametri di base sono:

- **url**: indirizzo al quale inviare la chiamata (default: pagina corrente)
- **success**: la funzione da chiamare nel caso la richiesta vada a buon fine
- **error**: la funzione da chiamare nel caso la richiesta fallisca
- **data**: i dati da passare al server via POST o GET
- **dataType**: la codifica dei dati inviati dal server (per noi sarà JSON)
- **asynch(true/false)**: se impostato a **false** si attende la risposta del server prima di proseguire con altre istruzione Javascript dopo la chiamata Ajax, se a **true** tutta la gestione avviene in asincrono (default: true)

Le richieste Ajax sul server sono sempre delle richieste HTTP, non cambia nulla rispetto a quelle inviate direttamente al browser; solo che questa volta restituiamo codice JSON e HTML.

Il controller deve smistare le richieste Ajax e creare delle viste che riempiano del JSON. Il modo più semplice per discriminare dalle altre richieste è usare un parametro.

### Restituire JSON da una Servlet

JSON è solo un formato diverso per una vista: non si restituisce codice HTML ma testo formattato in un altro modo. La soluzione più semplice è utilizzare una libreria di tag jsp per generare JSON: *JSON-taglib*.

La servlet richiamerà questa jsp come faceva quando doveva generare l'HTML.

#### JSON da una Servlet: Esempio (lato Servlet)

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // N.B. la servlet potrebbe gestire anche richieste che
    // prevedono la restituzione di codice HTML

    String command = request.getParameter("cmd");
    if (command != null) {
        if (command.equals("next") && request.getParameter("id") != null) {
            // devo produrre il JSON che descrive la prossima notizia
            int id = Integer.parseInt(request.getParameter("id"));
            News n = NewsFactory.getInstance().getNextNews(id);

            // imposto la news come attributo della request,
            // come facevamo per l'HTML
            request.setAttribute("news", n);

            // quando si restituisce del JSON e' importante segnalarlo ed
            // evitare il caching
            response.setContentType("application/json");
            response.setHeader("Expires", "Sat, 6 May 1995 12:00:00 GMT");
            response.setHeader("Cache-Control", "no-store, no-cache, must-
revalidate");

            // genero il JSON con una jsp
            request.getRequestDispatcher("newsJson.jsp").forward(request,
response);
        }
        // if per altri comandi...
    }
}
```

#### JSON da una Servlet: Esempio (lato JSP)

```
<%@page contentType="application/json" pageEncoding="UTF-8"%>
<%@taglib prefix="json" uri="http://www.atg.com/taglibs/json" %>
<json:object>

    <json:property name="id" value="${news.id}" />
    <json:property name="heading" value="${news.heading}" />
    <json:property name="img" value="${news.img}" />
    <json:property name="text" value="${news.text}" />
</json:object>
```

## JSON-tag lib

- json:object: rappresenta un oggetto JSON;
- json:array: rappresenta un array di oggetti JSON o sotto-array;
- json:property: rappresenta una proprietà di un oggetto (coppia nome-valore).

### JSON-tag: generare un array di notizie

```
<%@page contentType="application/json" pageEncoding="UTF-8"%>
<%@taglib prefix="json" uri="http://www.atg.com/taglibs/json" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<json:array>
    <c:forEach var="news" items="${newsList}">
        <json:object>
            <json:property name="id" value="${news.id}" />
            <json:property name="heading" value="${news.heading}" />
            <json:property name="img" value="${news.img}" />
            <json:property name="text" value="${news.text}" />
        </json:object>
    </c:forEach>
</json:array>
```

## Esempi di utilizzo di Ajax

Prima di effettuare la submit si fa una richiesta Ajax, che verifica sul server l'ammissibilità dei valori inseriti nei form, e a seconda del risultato della validazione può mostrare messaggi appositi senza ricaricare la pagina.

Inoltre permette di mostrare suggerimenti e utilizzare l'autocompletamento.

## Same Origin Policy

Per motivi di sicurezza l'interprete Javascript non permette di effettuare richieste Ajax verso origini diverse rispetto a quella della pagina attuale.

URL controllato	Risultato	Spiegazione
http://www.example.com/dir/page.html	Successo	Stesso protocollo e stesso host
http://www.example.com/dir2/other.html	Successo	Stesso protocollo e stesso host
http://www.example.com:81/dir/other.html	Fallimento	Stesso protocollo, stesso host ma porta diversa
https://www.example.com/dir/other.html	Fallimento	Protocollo diverso
http://en.example.com/dir/other.html	Fallimento	Host diverso
http://example.com/dir/other.html	Fallimento	Host diverso (devono essere esattamente uguali)
http://v2.www.example.com/dir/other.html	Fallimento	Host diverso (devono essere esattamente uguali)