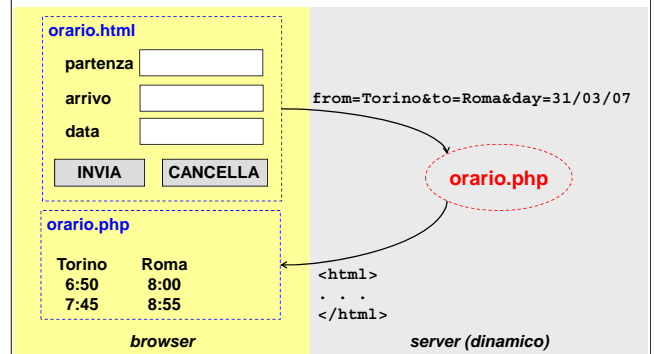


## I form HTML ed il web dinamico

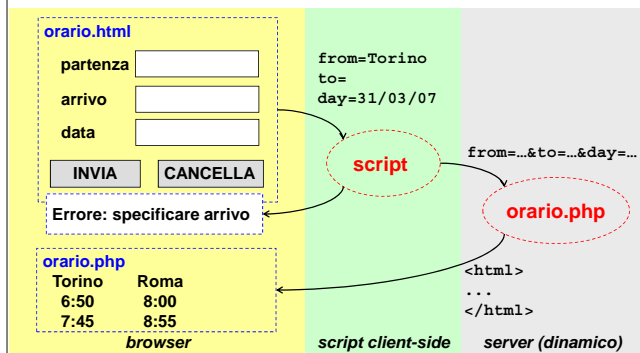
Antonio Lioy  
<lioy@polito.it>

Politecnico di Torino  
Dip. Automatica e Informatica

## Uso di form HTML per inviare dati nel web dinamico



## Accesso ai dati di un form HTML da uno script client-side



## Struttura di base dei form HTML

- **NAME** (o **ID**): nome simbolico per far riferimento al form
- **ACTION**: URL relativa a script CGI, PHP, ASP o a qualsiasi tipo di elaborazione sul server
- **METHOD**: GET oppure POST
- gli elementi di un form sono detti "controlli"

```
<form name="f1" method="get" action="http://...">
  <input ...>
  <select ...>
  ...
  <input type="submit" ...>
  <input type="reset" ...>
</form>
```

## Form: controlli di input

- tag **INPUT**
- **TYPE**: text, password, checkbox, radio, image, file, hidden, submit, reset, button
- **NAME**: nome simbolico (si può usare ID se è unico)
  - per passare i dati al server via HTTP
  - per accedere all'elemento da script client-side (es. JavaScript o VBScript)
- **VALUE**: contenuto iniziale del campo o valore da inviare

```
<input type=... name=... value=...>
```

## Form: pulsanti

- **SUBMIT**
  - creato tramite tag INPUT
  - invia i dati del form al server web
- **RESET**
  - creato tramite tag INPUT
  - imposta tutti controlli del form al valore di default
- **BUTTON**
  - creato tramite tag BUTTON
  - type=submit | reset | button
  - più "ricco" (es. testo + immagine) rispetto a INPUT SUBMIT/RESET ed usabile anche fuori da un form

### Form: controlli orientati al testo

- `<input type=text size=N maxlength=M name=...>`
  - zona di testo lunga N caratteri, al massimo M
- `<input type=password ...>`
  - come type=text ma visualizza i caratteri come \*
  - NON è un metodo sicuro per celare un password
- `<input type=hidden name=... value=...>`
  - valore fisso da trasmettere al server di nascosto
  - NON è un metodo sicuro per celare un dato
- `<textarea rows=NR cols=NC name=...>`  
... *testo iniziale* ...  
`</textarea>`
  - zona di testo di NR righe, ciascuna di NC caratteri

### Esempio di form (text, password)

```
<form name="f" action="/cgi/query" method="get">
<p> // nota: possibile usare (name) invece di (id)
your name: <input type="text" id="nome"> <br>
your home page:
<input type="text" id="home" value="http://"> <br>
password: <input type="password" id="pswd"> <br>
<input type="submit" value="ok">
<input type="reset" value="annulla">
</p>
</form>
```

### Interazione tra form e script

- script associato ad eventi DOM
  - `onclick = esegui_azione( );`
- all'interno dello script, i dati del form sono letti tramite il modello DOM in due modi diversi
  - gerarchia dei nomi  
`p = (document.f.pswd.value)`
  - estrazione diretta dell'elemento con ID unico  
`p = getElementById("pswd").value`

### Interazione form-script (name)

```
<script type="text/javascript">
function display(x) {
  alert ("nome: " + x.nome.value)
  alert ("home page: " + x.home.value)
  alert ("password: " + x.pswd.value)
}
</script>
. . .
<form name="f" action="/cgi/query" method="get"
  onsubmit="display(f)">
```

uhp\_name.html

### Interazione form-script (id)

```
<script type="text/javascript">
function display( ) {
  n = document.getElementById("nome").value
  h = document.getElementById("home").value
  p = document.getElementById("pswd").value
  alert ("nome: " + n)
  alert ("home page: " + h)
  alert ("password: " + p)
}
</script>
. . .
<form name="f" action="/cgi/query" method="get"
  onsubmit="display()">
```

uhp\_id.html

### Form: menù

- tag SELECT per racchiudere le varie opzioni
- tag OPTION per le singole opzioni
- tag OPTGROUP per raggruppare opzioni (menù a cascata; un solo livello di raggruppamento)
- attributo LABEL (vince rispetto al testo del tag)
- attributo VALUE (se manca, valore uguale testo)

```
<select name=...>
  <option label=... >
  <option label=... value=... >
  <option> ... </option>
  <option value=... > ... </option>
</select>
```

### Form: opzioni dei menù

- attributo **SELECTED** (su **OPTION**) per opzione pre-selezionata
  - default: dipende dallo UA
- attributo **MULTIPLE** (su **SELECT**) per indicare che sono possibili scelte multiple
  - default: scelta singola

```
<select multiple name=...>
  <option label=... >
  <option> ... </option>
  <option selected> ... </option>
</select>
```

### Form: controlli a scelta multipla

- **CHECKBOX**
  - un elemento di tipo on/off
  - indipendente da altri controlli dello stesso tipo
  - inviati al server tutti quelli selezionati (CHECKED)
  - potrebbe anche non inviare niente al server
- **RADIO**
  - un insieme di elementi di tipo on/off identificati dallo stesso NAME (in questo caso NON si può usare ID perché il nome non è unico ma comune)
  - mutuamente esclusivi (selezionabili uno solo)
  - inviato al server il valore dell'unico elemento selezionato (CHECKED)

### Esempio di form (checkbox)

```
<form action="/cgi/query" method="get">
<p>
Compose your own fruit salad:
<br>
<input type="checkbox" id="banana"> Banana
<input type="checkbox" id="apple" checked> Apple
<input type="checkbox" id="orange"> Orange (red)
<br>
<input type="submit">
<input type="reset">
</p>
</form>
```

Si noti:  
 - id diverso per ogni pulsante  
 - pulsante pre-selezionato (checked)

### Esempio di form (radio)

```
<form name="f" action="/cgi/query" method="get">
<p>
Select your preferred fruit:
<input type="radio" name="frt" value="banana">
Banana <br>
<input type="radio" name="frt" value="apple"
checked>
Apple <br>
<input type="radio" name="frt" value="orange">
Orange (red) <br>
<input type="submit">
<input type="reset">
</p>
</form>
```

Si noti:  
 - stesso name per i tre pulsanti  
 - valore di default (checked)

### Controlli disabilitati o a sola lettura

- attributo **"readonly"**
  - non permette all'utente di cambiare il valore di un controllo (possibile solo tramite script client-side)
  - valido nei controlli INPUT e TEXTAREA
- attributo **"disabled"**
  - disabilita un controllo
    - l'utente non può cambiarne il valore
    - non verrà inviato al server
  - valido nei controlli INPUT, TEXTAREA, BUTTON, SELECT, OPTION, OPTGROUP
- attributi **Booleani**, cambiabili da script client-side

### Uso di script per validare un form

- si usa il modello degli eventi DOM per attivare uno script lato client
- tipicamente si associa uno script all'evento **onSubmit**:
  - script eseguito quando si preme il pulsante Submit
  - se lo script restituisce il valore "true" allora i dati del form vengono inviati al server, altrimenti no
- possibile associare script ad altri eventi (es. **onChange**) per validare singoli controlli non appena vengono introdotti dei dati

### Esempio di script per validare un form

```
<form name="sample" method="post" action="..."
  onSubmit="return validateForm()">
<p>
Nome:
  <input type="text" name="nome" size="30"><br>
Età:
  <input type="text" name="eta" size="3"><br>
Data di nascita:
  <input type="text" name="nascita" size="10"><br>
  <input type="submit">
  <input type="reset">
</p>
</form>
```

### Script di validazione

```
<script type="text/javascript">
function validateForm()
{
  formObj = document.sample;
  if (formObj.nome.value == "") {
    alert("Non hai introdotto il nome!");
    return false;
  }
  else if (formObj.eta.value == "") {
    alert("Non hai introdotto l'età!");
    return false;
  }
  else if ... return false;
  return true;
}
</script>
```

### Come fare la validazione?

- **controllare che il valore associato ad un controllo:**
  - non sia vuoto (se è un caso possibile dato il tipo)
  - abbia un valore corretto ("looks good") piuttosto che non abbia un valore sbagliato ("doesn't look bad")
- **esempio (validazione del valore di un controllo di testo usato per introdurre un CAP):**
  - contenga solo caratteri numerici ('0'...'9')
  - sia composto esattamente da cinque caratteri
  - (opzionale) se è noto l'elenco di tutti i possibili CAP controllare che il valore fornito sia uno di questi
- **in caso di errore, fornire un avviso all'utente che lo aiuti a correggere l'errore (ossia NON "CAP errato")**

### Trasmissione parametri di un form (GET)

- **URI = concatenazione del campo "action" con "?" e quindi i parametri espressi nella codifica [application/x-www-form-urlencoded](#)**
- **il body HTTP della richiesta rimane vuoto**

### application/x-www-form-urlencoded

- è la codifica di default
- usabile sia con GET sia con POST
- genera una stringa composta dai nomi dei controlli del form seguiti da "=" e dai valori inseriti:
   
`nome_ctrl1=val_ctrl1&nome_ctrl2=val_ctrl2&...`
- **separatore tra un controllo ed il successivo: &**
- **spazi nei nomi o nei valori sostituiti da "+"**
- **caratteri speciali, non US-ASCII o con significato particolare (/ ? ...) sostituiti con %xx (ove "xx" è il numero esadecimale del suo codice ISO-8859-1)**

### Es. x-www-form-urlencoded: form

```
<form
  name="sample"
  method="get"
  action="/cgi/acquisisci">
Nome e cognome:
  <input type="text" name="cognome" size="30"><br>
Numero di figli:
  <input type="text" name="figli" size="3"><br>
Data di nascita:
  <input type="text" name="nascita" size="10"><br>
  <input type="submit">
  <input type="reset">
</form>
```

**Es. x-www-form-urlencoded: trasmissione**

- se il precedente form venisse riempito dal signor Marco Noè, nato il 30/10/74, genitore di 3 figli ...
- ... allora verrebbe creata la seguente stringa:

```
cognome=Marco+Noè&figli=3&nascita=30%2F10%2F74
```

**Trasmissione parametri di un form (POST)**

- URI coincide col valore del campo "action"
- (default, ovvero enctype non specificato)
  - Content-Type: application/x-www-form-urlencoded
  - Content-Length: ...
  - body contiene solo la stringa dei parametri nella forma application/x-www-form-urlencoded
- con "enctype=multipart/form-data"
  - Content-Type: multipart/form-data
  - Content-Length: ...
  - body = messaggio MIME (una sezione per ogni parametro)
  - nota: obbligatorio per controlli di tipo File

**Esempio invio dati con GET: il form**

```
<form method="get" action="/cgi/insaula">
<table border="0">
<tr>
<td>Numero aula:</td>
<td><input type="text" size="8" name="num"></td>
</tr>
<tr>
<td>Sede:</td>
<td><input type="text" size="15" name="sede"></td>
</tr>
<tr>
<td>
<input type="submit" value="Invia">
<input type="reset" value="Cancella">
</td>
</tr>
</table>
</form>
```

**Esempio invio dati con GET: URI, HTTP, env**

URI  
http://localhost/cgi/insaula?num=12A&sede=Sede+Centrale

canale HTTP (C > S)  
GET /cgi/insaula?num=12A&sede=Sede+Centrale HTTP/1.1  
Accept: image/gif, image/x-bitmap, image/jpeg, /\*  
Referer: http://localhost/pad/formaula.html  
Accept-Language: it  
Accept-Encoding: gzip, deflate  
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)  
Host: 192.168.235.10  
Connection: Keep-Alive

QUERY\_STRING  
num=12A&sede=Sede+Centrale

**Invio dati con POST (caso 1): il form**

```
<form method="post" action="/cgi/insaula">
<table border="0">
<tr>
<td>Numero aula:</td>
<td><input type="text" size="8" name="num"></td>
</tr>
<tr>
<td>Sede:</td>
<td><input type="text" size="15" name="sede"></td>
</tr>
<tr>
<td>
<input type="submit" value="Invia">
<input type="reset" value="Cancella">
</td>
</tr>
</table>
</form>
```

**Invio dati con POST (caso 1): URI, HTTP, env**

URI  
http://localhost/cgi/insaula

canale HTTP (C > S)  
POST /cgi/insaula HTTP/1.1  
Accept: image/gif, image/x-bitmap, image/jpeg, /\*  
Referer: http://localhost/pad/formaula.html  
Accept-Language: it  
Accept-Encoding: gzip, deflate  
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)  
Host: 192.168.235.10  
Connection: Keep-Alive  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 23

QUERY\_STRING  
num=12A&sede=Sede+Centrale

### Invio dati con POST (caso 2): il form

```
<form method="post" action="/cgi/insaula"
  enctype="multipart/form-data">
<table border="0">
<tr>
  <td>Numero aula:</td>
  <td><input type="text" size="8" name="num"></td>
</tr>
<tr>
  <td>Sede:</td>
  <td><input type="text" size="15" name="sede"></td>
</tr>
<tr>
  <td>
    <input type="submit" value="Invia">
    <input type="reset" value="Cancella">
  </td>
</tr>
</table>
</form>
```

### Invio dati con POST (caso 2): URI, HTTP, env

URI  
http://localhost/cgi/insaula

canale HTTP (C > S)

```
POST /cgi/insaula HTTP/1.1
. . .
Content-Type: multipart/form-data; boundary=AaBbCc
Content-Length: 145

--AaBbCc
Content-Disposition: form-data; name="num"

12A
--AaBbCc
Content-Disposition: form-data; name="sede"

Sede Centrale ← N.B. valore non codificato
--AaBbCc--
```

### Attenzione ai campi vuoti !

- ad eccezione dei controlli SELECT e RADIO ...
- ... tutti gli altri campi di un form possono non trasmettere input
- ... ed in un caso (TYPE=CHECKBOX) può anche non essere presente la variabile relativa al campo (se è OFF)
- le applicazioni che ricevono input da un form devono saper trattare tutti i casi

### Esempio 1

```
<form name="sample" method="get"
  action="http://www.negozio.it/pagamento.asp">
Carta di credito:
<input type="text" name="cardno" size="16">
<br>
MasterCard
<input type="radio" name="cc" value="mastercard">
<br>
Visa
<input type="radio" name="cc" value="visacard">
<br>
<input type="submit"> <input type="reset">
</form>
```

cardno=123456789012345&cc=mastercard

### Esempio 2: il form

```
<form name="sample" method="get"
  action="http://www.amici.it/persona.asp">
cognome: <input type="text" name="cogn" size="30">
<br>
hobby:
<ul>
<li>pesca <input type="checkbox" name="cb_pesca">
<li>sci <input type="checkbox" name="cb_sci">
</ul>
<input type="submit"> <input type="reset">
</form>
```

### Esempio 2: dati trasmessi al server

L'applicazione lato server deve gestire questi casi (ed altre combinazioni ...)

cogn=

cogn=De+Chirico

cogn=De+Chirico&cb\_pesca=on

cogn=De+Chirico&cb\_pesca=on&cb\_sci=on

### Form: file upload

- il controllo `<input type="file" ...>` inserisce un elemento per la selezione del nome di un file
- la forma esatta del controllo dipende dal browser ma spesso:
  - campo di testo per inserire direttamente il nome
  - pulsante per attivare interfaccia grafica (navigazione del file system locale e selezione del file)
- tutti i dati del form sono trasmessi singolarmente come parti di un messaggio MIME
- usabile solo con POST e tipo MIME specifico:

```
<form action=...
  enctype="multipart/form-data" method="post">
```

### Esempio di form (file upload)

```
<form
  action="/cgi/fileprint"
  enctype="multipart/form-data"
  method="post">
  File da stampare:
  <input type="file" name="myfile">
  Numero di copie da stampare:
  <input type="text" name="ncopie" size="2">
  <br><br>
  <input type="submit" value="submit">
</form>
```

### File upload – trasferimento C > S

```
POST /cgi/fileprint HTTP/1.1
Host: server.it
Content-Type: multipart/form-data; boundary=AaBb
Content-Length: 199

--AaBb
Content-Disposition: form-data; name="myfile";
  filename="orario.txt"
Content-Type: text/plain

8:30-12:30 aula 12
--AaBb
Content-Disposition: form-data; name="ncopie"

3
--AaBb--
```

### Form: meglio trasmissione con GET o POST?

- **GET:**
  - permette di fare caching della pagina di risposta
  - permette di creare bookmark e link alla pagina
  - lascia traccia del valore dei parametri nel log del server (problema di privacy e/o sicurezza)
  - alcuni server limitano la lunghezza della query string a 256 caratteri se è all'interno della URI
  - semplifica il debug dei form
- **POST:**
  - non permette caching e bookmarking
  - non lascia traccia nel log
  - non pone limiti alla query string