

Tecnologie per la Comunicazione Aziendale

Flavio De Paoli
depaoli@disco.unimib.it



Applicazioni Web - Outline

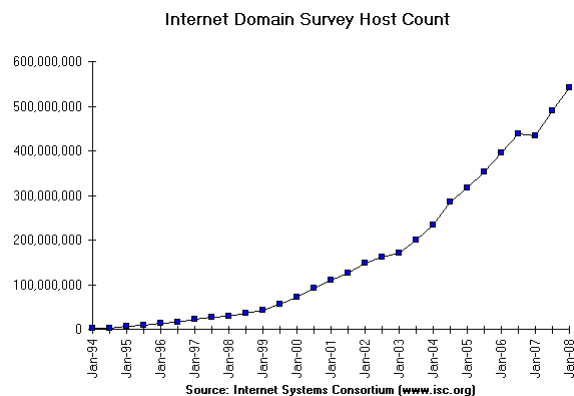
- Il web come architettura di riferimento
- Architettura di una applicazione web
- Servlet e JSP
- Gestione delle sessioni: cookies
- Accesso a database con una servlet
- Pagine JSP



Perché il web

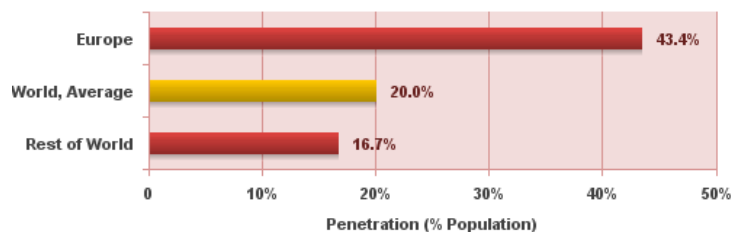
- **Basato su Internet**
 - ✓ Ambiente standard (TCP/IP)
 - ✓ Larga diffusione
 - ✓ Indipendente dalle piattaforme
- **Semplicità d'uso**
 - ✓ Interfaccia grafica (Browsers)
- **Infrastruttura completa**
 - ✓ Supporta sistemi aperti
 - ✓ Strumenti sempre più potenti: evoluzioni di HTML, CGI, JavaScript, Java, ...

Diffusione di Internet



Diffusione di internet

Internet Penetration in Europe December 2007

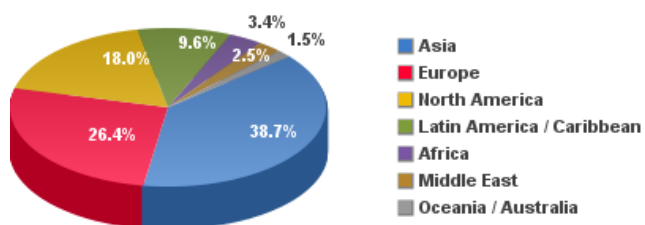


Source: www.internetworldstats.com
Copyright © 2008, Miniwatts Marketing Group

Persone on-line

➤ Internet users: 1.319.872.109 [28.04.2008]

World Internet Users December 2007



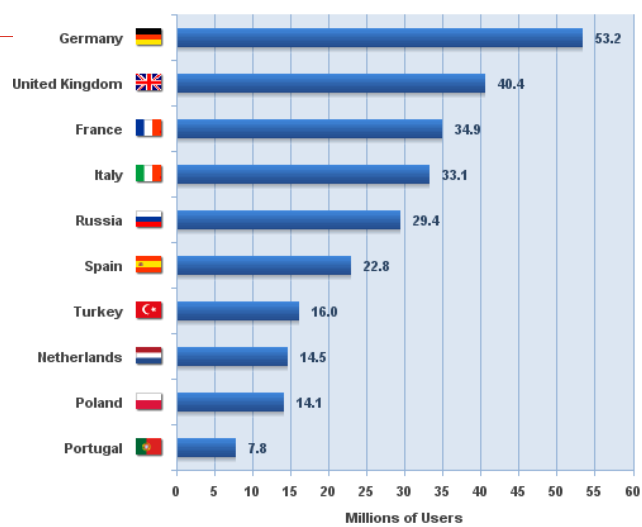
Source: www.internetworldstats.com
Copyright © 2008, Miniwatts Marketing Group

Utilizzo

➤ Principali attività in rete [Ofcom, Q1, 2005]



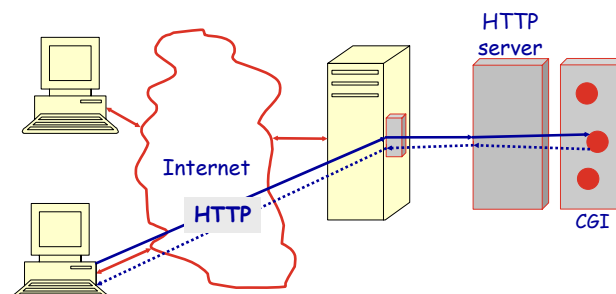
Top Ten Internet Countries in Europe December 2007



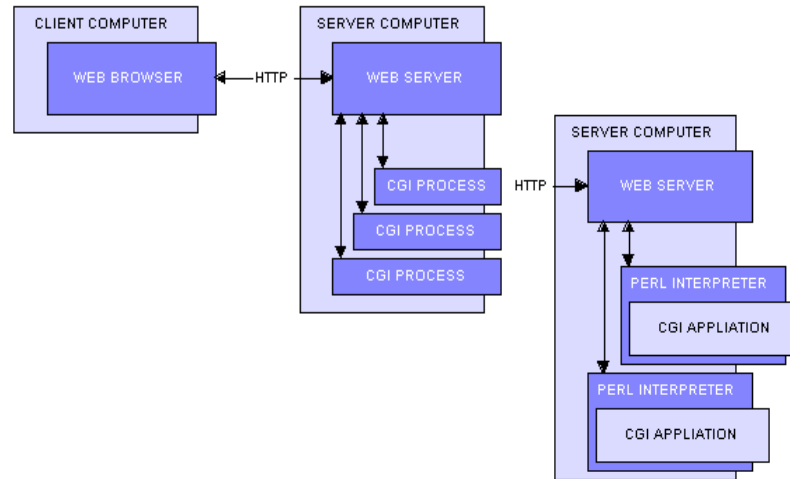
HTTP/CGI

- URL definisce un naming globale
- HyperText Transfer Protocol (HTTP)
 - ✓ Protocollo di tipo client/server che permette scambio di messaggi in formato testuale – basato su TCP/IP
 - ✓ Può essere utilizzato per incapsulare invocazioni di programmi sul server con passaggio parametri (dati utilizzati dal programma)
 - ✓ I risultati sono trasmessi come pagine HTML
- Common Gateway Interface (CGI)
 - ✓ Definisce le caratteristiche dei programmi lato server
 - ✓ Definisce le modalità con cui i dati vengono trasferiti tra Web Servers e programmi

Architettura web app



Architettura CGI



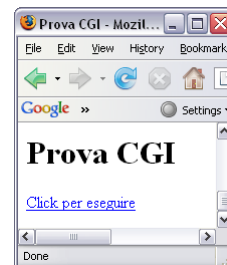
CGI lato Client

- La pagina web deve contenere il riferimento allo script

```
<h1> Prova CGI</h1>
<p>
<a href="http://149.132.196.196/Scripts/nome">
  Click per eseguire
</a>
</p>
```

- Il browser invia una richiesta tipo:

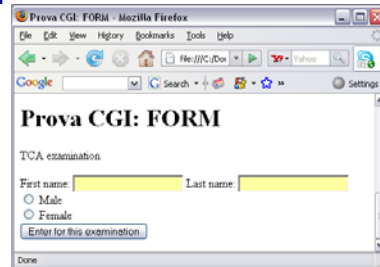
```
GET /Scripts/nome HTTP/1.0
```



CGI e Form

- > Si possono usare le form per inviare dati

```
...
<h1> Prova CGI: FORM </h1>
<p>TCA examination
<br>
<form action="..." method="post">
First name:
<input type="text" name="firstname">
Last name:
<input type="text" name="lastname">
<br>
<input type="radio" name="sex" value="male"> Male
<br>
<input type="radio" name="sex" value="female"> Female
<br>
<input type="submit" value="Enter for this examination">
</form>
</p>
...
```



CGI e form

- > Il client puo' usare uno dei due metodi:

- ✓ GET

Include l'input in coda alla URL

E' idempotente: si assume che ogni esecuzione abbia lo stesso effetto => la risposta viene conservata nella cache del client

Usata per ottenere pagine html e immagini

- ✓ POST

L'input segue come documento autonomo

Non e' idempotente: si assume che ogni esecuzione abbia un diverso effetto => La risposta NON viene conservata nella cache

Usata per processare FORM e interagire con DB

- ✓ Esempio

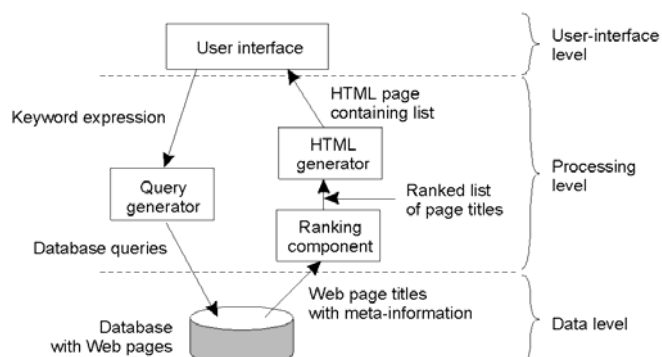
```
GET Scripts/nomeScript/Jserver="pippo.pluto.it"
/inviaRisposta="false" HTTP/1.0
```

Caratteristiche

- **Limiti del protocollo HTTP a caratteri**
 - ✓ Utilizzo dell'interfaccia browser: uso di FORM
 - ✓ Lentezza: occorre tradurre e ritradurre i dati (es. da testo a numeri: da "256" a 256)
 - ✓ Costruzione delle pagine HTML in risposta
- **Mancanza di stato**
 - ✓ Ogni richiesta è un messaggio autonomo
 - ✓ Per creare sessioni di lavoro (legare piu' richieste tra loro) occorre includere informazioni con mezzi espliciti
 - campi nascosti
 - cookie

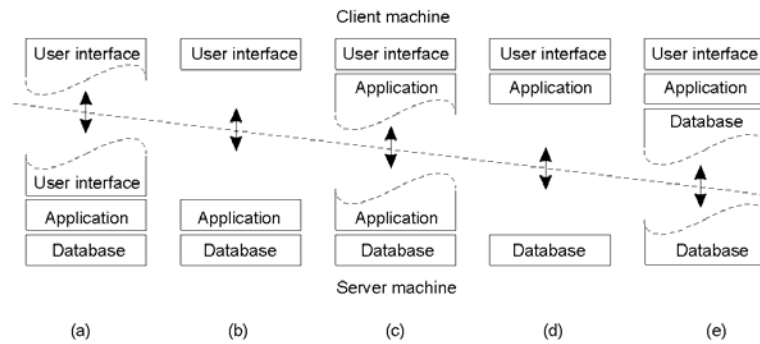
Come è fatta un'applicazione

- **Un esempio di motore di ricerca organizzato a tre livelli**



Architetture multilivello

➤ Alternative per applicazioni client-server (a) – (e)



➤ Le applicazioni Web sono di tipo (a) – (c)

Java Servlet

Caratteristiche

- Sono applicazioni lato server
- Hanno una interfaccia standard
 - ✓ sono limitate: non posso fare ciò che voglio
 - ✓ sono semplici: basta realizzare le parti mancanti
- Sono residenti in memoria
 - ✓ mantengono uno stato: posso realizzare delle sessioni
 - ✓ consentono interfaccia con un altre servlet
- Vantaggi
 - ✓ sono più efficienti e potenti delle CGI in quanto utilizzano un linguaggio e un ambiente completo (Java)
 - ✓ consentono di integrare sistemi e applicazioni dando accesso via Web

Cos'è e come funziona

- Una servlet è un *componente* gestito in modo automatico da un *container* (o *engine*)
- L'interfaccia definisce un set di operazioni predichiarate e (ri)definibili di volta in volta
- Il container controlla le servlet (attiva/disattiva) in base alle richieste dei client

Interfaccia Servlet

- Ogni servlet implementa l'interfaccia `javax.servlet.Servlet`, con 5 metodi
 - ✓ void **init**(ServletConfig config)
Inizializza la servlet
 - ✓ void **destroy**()
Chiamata quando la servlet termina (es: per chiudere un file o una connessione con un database)
 - ✓ ServletConfig **getServletConfig**()
Restituisce i parametri di inizializzazione e il ServletContext che da accesso all'ambiente
 - ✓ String **getServletInfo**()
Restituisce informazioni tipo autore e versione
 - ✓ void **service**(ServletRequest request, ServletResponse response)
Invocato per gestire le richieste dei client

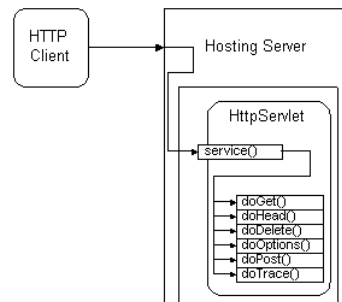
Classi astratte

- L'interfaccia è solo la dichiarazione dei metodi che, per essere utilizzabili, devono essere implementati in una classe (implementare = associare il codice, cioè le istruzioni, che definiscono il compito assegnato al metodo)
- Sono presenti due classi *astratte*, cioè che implementano i metodi dell'interfaccia in modo che non facciano nulla.
 - ✓ `javax.servlet.GenericServlet`
 - ✓ `javax.servlet.http.HttpServlet`
Definisce metodi per l'uso in ambiente web
- Questo semplifica la scrittura delle servlet vere e proprie in quando basta implementare (ridefinendoli) solo i metodi che interessano

La classe `HttpServlet`

- Implementa **service** in modo da legare le invocazioni al protocollo HTTP, cioè ai tipi GET, POST, ...

- ✓ `doGet`
Processa le richieste di tipo GET
- ✓ `doPost`
Processa le richieste di tipo POST
- ✓ Altri minori ...
- ✓ Parametri:
`HttpServletRequest`
`HttpServletResponse`
- ✓ Eccezioni
`ServletException`
`IOException`



Richieste e risposte

- Anche i parametri sono stati adattati al protocollo HTTP, cioè consentono di costruire dei messaggi HTTP specificando i dati da inserire nell'head e nel body di un messaggio.
- Interfaccia **`HttpServletRequest`**
 - ✓ Viene passato un oggetto da `service`
 - ✓ Contiene la richiesta del client
 - ✓ Estende `ServletRequest`
- Interfaccia **`HttpServletResponse`**
 - ✓ Viene passato un oggetto da `service`
 - ✓ Contiene la risposta per il client
 - ✓ Estende `ServletResponse`

Richieste e risposte

➤ I metodi

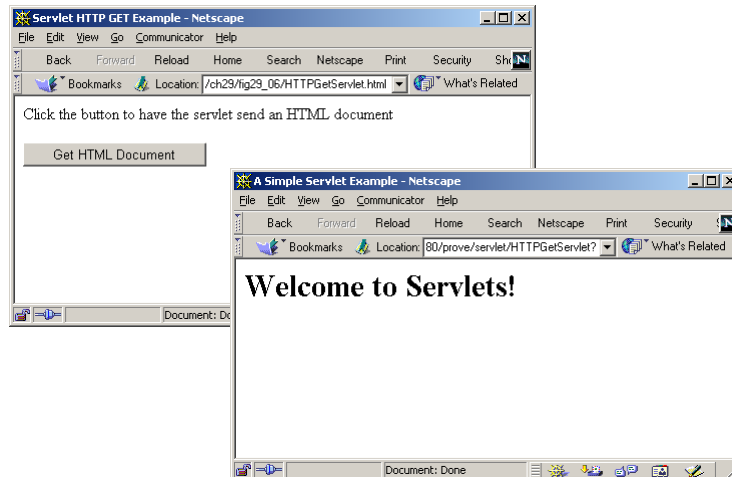
- ✓ String **getParameter**(String name)
Restituisce il valore dell'argomento name
- ✓ Enumeration **getParameterNames**()
Restituisce l'elenco dei nomi degli argomenti
- ✓ String[] **getParameterValues**(String name)
Restituisce i valori dell'argomento name
- ✓ Cookie[] **getCookies**()
Restituisce i cookies del server sul client
- ✓ void **addCookie**(Cookie cookie)
Aggiunge un cookie nell'intestazione della risposta
- ✓ HttpSession **getSession**(boolean create)
Una HttpSession identifica il client.
Viene creata se create=true

Richieste e risposte

➤ I metodi (continua)

- ✓ void **setContentType**(String type)
Specifica il tipo MIME della risposta per dire
al browser come visualizzare la risposta
Es: "text/html" dice che e' html
- ✓ ServletOutputStream **getOutputStream**()
Restituisce lo stream di byte per comunicare con
il client
- ✓ PrintWriter **getWriter**()
Restituisce lo stream di caratteri per
comunicare con il client

Un esempio GET



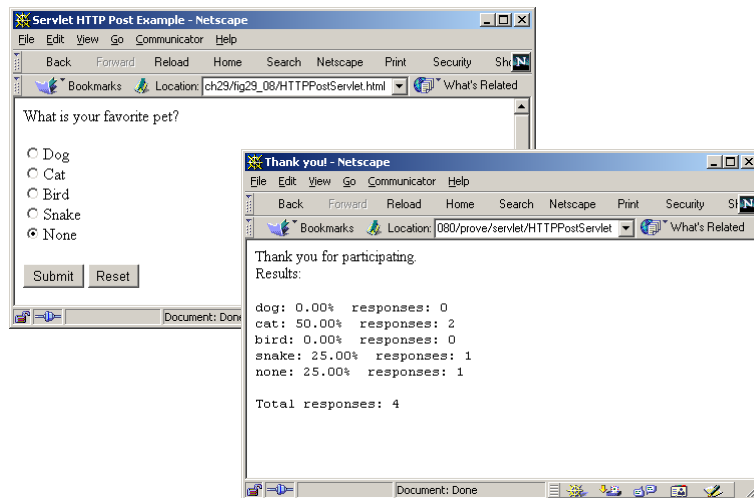
Lato client: la pagina HTML

- 1) `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">`
 - 2) `<HTML>`
 - 3) `<HEAD>`
 - 4) `<TITLE>Servlet HTTP GET Example</TITLE>`
 - 5) `</HEAD>`
 - 6) `<BODY>`
 - 7) `<FORM`
`ACTION="http://localhost:8080/servlet/HTTPGetServlet"`
 - 8) `METHOD="GET">`
 - 9) `<P>Click the button to have the servlet send`
`an HTML document</P>`
 - 10) `<INPUT TYPE="submit" VALUE="Get HTML Document">`
 - 11) `</FORM>`
 - 12) `</BODY>`
 - 13) `</HTML>`
 - 14) `</HTML>`
- Annotations in the image:
- "Come processare il form" points to the `ACTION` attribute.
 - "Tipo di richiesta" points to the `METHOD="GET"` attribute.
 - "Crea un bottone" points to the `VALUE="Get HTML Document"` attribute.
 - "Etichetta esposta" points to the `TYPE="submit"` attribute.

Lato server: il metodo doGet

```
1) // da Internet e WWW - How to program, Dietel&Dietel, Prentice Hall
2) // Creating and sending a page to the client
3) public class HTTPGetServlet extends HttpServlet {
4)     public void doGet( HttpServletRequest request,
5)                       HttpServletResponse response )
6)         throws ServletException, IOException {
7)         PrintWriter output;
8)         response.setContentType( "text/html" ); // content type
9)         output = response.getWriter(); // get writer
10)        // create and send HTML page to client
11)        StringBuffer buf = new StringBuffer();
12)        buf.append( "<HTML><HEAD><TITLE>\n" );
13)        buf.append( "A Simple Servlet Example\n" );
14)        buf.append( "</TITLE></HEAD><BODY>\n" );
15)        buf.append( "<H1>Welcome to Servlets!</H1>\n" );
16)        buf.append( "</BODY></HTML>" );
17)        output.println( buf.toString() );
18)        output.close(); // close PrintWriter stream
19)    }
20) }
```

Un esempio POST



The screenshot shows two Netscape browser windows. The main window, titled "Servlet HTTP Post Example - Netscape", displays a form with the question "What is your favorite pet?". The form has radio buttons for "Dog", "Cat", "Bird", "Snake", and "None", with "None" selected. There are "Submit" and "Reset" buttons. The location bar shows "ch29/fig29_08/HTTPPostServlet.html".

The second window, titled "Thank you - Netscape", shows the results of the POST request. It displays "Thank you for participating." followed by a table of results:

Pet	Percentage	Count
dog	0.00%	0
cat	50.00%	2
bird	0.00%	0
snake	25.00%	1
none	25.00%	1
Total responses:		4

The location bar of the second window shows "080/prove/servlet/HTTPPostServlet".

Lato client: la pagina HTML

```
1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2. <HTML>
3. <HEAD>
4. <TITLE>Servlet HTTP Post Example</TITLE>
5. </HEAD>
6. <BODY>
7. <FORM ACTION="http://localhost:8080/servlet/HTTPPostServlet"
8.     METHOD="POST">
9.     What is your favorite pet?<BR><BR>
10.    <INPUT TYPE="radio" NAME="animal" VALUE="dog">Dog<BR>
11.    <INPUT TYPE="radio" NAME="animal" VALUE="cat">Cat<BR>
12.    <INPUT TYPE="radio" NAME="animal" VALUE="bird">Bird<BR>
13.    <INPUT TYPE="radio" NAME="animal" VALUE="snake">Snake<BR>
14.    <INPUT TYPE="radio" NAME="animal" VALUE="none"
15.        CHECKED>None
16.    <BR><BR><INPUT TYPE="submit" VALUE="Submit">
17.    <INPUT TYPE="reset">
18. </FORM>
19. </BODY>
20. </HTML>
```

Lato server: il metodo doPost

```
1) public class HTTPPostServlet extends HttpServlet {
2)     // definisco l'elenco degli animali
3)     private String animalNames[] =
4)         { "dog", "cat", "bird", "snake", "none" };
5)
6)     public void doPost( HttpServletRequest request,
7)                         HttpServletResponse response )
8)     {
9)         int animals[] = null, // contatori di preferenze
10)        total = 0; // totale delle preferenze espresse
11)        // i dati sono memorizzati nel file "survey.dat"
12)        File f = new File("survey.dat"); // apro o creo il file
13)        if ( f.exists() ) {
14)            // leggo il file e lo assegno alla variabile animals
15)            animals = ...
16)            // conto quante sono le risposte date in precedenza
17)            for ( int i = 0; i < animals.length; ++i )
18)                total += animals[ i ];
19)        }
20)        else // creo un nuovo array di contatori
21)            animals = new int[ 5 ];
22)    }
23)
24)
```

```

25) // leggo il messaggio con la nuova preferenza
26) String value = request.getParameter( "animal" );
27) ++total; // aggiorno il totale delle risposte

37) // determino quello votato e aggiorno il suo contatore
38) for ( int i = 0; i < animalNames.length; ++i )
39)     if ( value.equals( animalNames[ i ] ) )
40)         ++animals[ i ];

41) // scrivo i nuovi contatori sul file e lo chiudo
42) ObjectOutputStream output = new ObjectOutputStream(
43)     new FileOutputStream( f ) );

44) output.writeObject( animals );
45) output.flush();
46) output.close();

47) // calcolo le percentuali
48) double percentages[] = new double[ animals.length ];
49)
50) for ( int i = 0; i < percentages.length; ++i )
51)     percentages[ i ] = 100.0 * animals[ i ] / total;

```

33

```

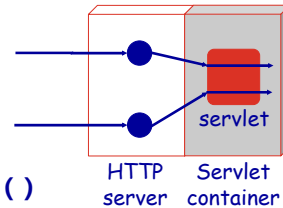
52) // costruisco l'head del messaggio di risposta
53) response.setContentType( "text/html" ); // content type
54) // predispongo alla scrittura del body del messaggio
55) PrintWriter responseOutput = response.getWriter();
56) // uso un Buffer di servizio per costruire la pagina
57) StringBuffer buf = new StringBuffer();
58) buf.append( "<html>\n" );
59) buf.append( "<title>Thank you!</title>\n" );
60) buf.append( "Thank you for participating.\n" );
61) buf.append( "<BR>Results:\n<PRE>" );
62) DecimalFormat twoDigits = new DecimalFormat( "#0.00" );
63) for ( int i = 0; i < percentages.length; ++i ) {
64)     buf.append( "<BR>" );
65)     buf.append( animalNames[ i ] );
66)     buf.append( ": " );
67)     buf.append( twoDigits.format( percentages[ i ] ) );
68)     buf.append( "% responses: " );
69)     buf.append( animals[ i ] );
70)     buf.append( "\n" );
71) }
72) buf.append( "\n<BR><BR>Total responses: " );
73) buf.append( total );
74) buf.append( "</PRE>\n</html>" );
75) // scrivo la pagina nella risposta
76) responseOutput.println( buf.toString() );
77) responseOutput.close();
78) }
79) }

```

34

Ciclo di vita

- Una servlet viene creata dal container
 - ✓ Quando viene effettuata la prima chiamata
- Viene invocato il metodo `init()` per inizializzazioni specifiche
- Una servlet viene distrutta
 - ✓ Quando non ci sono servizi in esecuzione
 - ✓ Quando e' scaduto un timeout predefinito
- Viene invocato il metodo `destroy()` per terminare correttamente la servlet



Le sessioni

- HTTP non prevede persistenza
 - ✓ Non si possono mantenere informazioni tra una chiamata e l'altra
 - ✓ Non si possono identificare i clienti
- Cookies
 - ✓ Informazioni memorizzate sul client
 - ✓ Permettono di gestire sessioni
- HttpSession
 - ✓ Gestito automaticamente dal container (con cookie o riscrittura delle URL)

Cookie

- Stringhe di caratteri restituite dal server insieme alle risposte
- Inserirle nell'header del messaggio HTTP in risposta
 - ✓ Vengono memorizzati
nome, valore e host di provenienza
tempo di validità (fino a maxCookie)

User-server interaction: cookies

- server sends "cookie" to client in response msg
`Set-cookie: 1678453`
 - client presents cookie in later requests
`cookie: 1678453`
 - server matches presented-cookie with server-stored info
 - ✓ authentication
 - ✓ remembering user preferences, previous choices
-
- ```
graph LR
 subgraph Scenario1 [Scenario 1: Initial Cookie Setting]
 C1[client] -- "usual http request msg" --> S1[server]
 S1 -- "usual http response + Set-cookie: #" --> C1
 end
 subgraph Scenario2 [Scenario 2: Cookie Usage]
 C2[client] -- "usual http request msg
cookie: #" --> S2[server]
 S2 -- "usual http response msg" --> C2
 end
 S1 -.-> SA1[cookie-specific action]
 S2 -.-> SA2[cookie-specific action]
```

## Definire un Cookie

```
// creare un cookie
Cookie nomeCookie = new Cookie("nome", valore);
// definire gli attributi
nomeCookie.setComment("Questo e' un commento " +
 "al cookie.");
nomeCookie.setMaxAge(nSecondi);
// inviare il cookie con la risposta
// NOTA: deve essere aggiunta alla risposta prima
// dei dati (prima di invocare getWriter)
response.add(nomeCookie);
```

## Recuperare un Cookie

```
Cookie[] cookies = request.getCookies();
for(i=0; i < cookies.length; i++) {
 Cookie thisCookie = cookie[i];
 if (thisCookie.getName().equals("nome") &&
 thisCookie.getValue().equals(valore)) {
 ...
 // cancella il cookie mettendo l'eta' a 0
 thisCookie.setMaxAge(0);
 }
}
```

# Java JSP

Flavio De Paoli



## Outline

- Cos'è una JSP
- Gli elementi che compongono una JSP
- JSP e JavaBeans



## Java Server Pages

- Tecnologia per la creazione di applicazioni web
- Specifica l'interazione tra un contenitore/server ed un insieme di "pagine" che presentano informazioni all'utente
- Le pagine sono costituite da tag tradizionali (HTML, XML, WML, ...) e da tag applicativi che controllano la generazione del contenuto
- Rispetto ai servlet, facilitano la separazione tra logica applicativa e presentazione
  
- Analogo alla tecnologia Microsoft Active Server Page (ASP)
- Differenze
  - ✓ una Java Server Page chiama un programma Java eseguito sul Web server
  - ✓ una Active Server Page contiene uno script VBScript o JScript

## Java Server Pages

- JavaServer Pages (JSP) separano la parte dinamica delle pagine dal template HTML statico
  - ✓ Il codice JSP va incluso in tag speciali, delimitati da "<%>" e "%>".
- Esempio
  - ✓ una pagina che visualizza  
"Grazie per la scelta di *Internet Guida Pratica*"  
quando l'utente si connette all'URL  
`http://host/OrderConfirmation.jsp?title=Internet+Guida+Pratica`
  - ✓ contiene  
Grazie per la scelta di `<I><%= request.getParameter("title") %>`  
`</I>`
- La pagina viene convertita automaticamente in una servlet java la prima volta che viene richiesta

## JSP: esempio

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Uso di JSP</TITLE>
<LINK REL=STYLESHEET
 HREF="My-Style-Sheet.css"
 TYPE="text/css">
</HEAD>
<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE"
 VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
<TABLE BORDER=5 BGCOLOR="#EF8429">
 <TR><TH CLASS="TITLE">
 Using JavaServer Pages</TH>
</TR>
</TABLE>
</CENTER>
<P>
Some dynamic content created using various JSP mechanisms:

Expression.

Your hostname: <%= Request.getRemoteHost() %>.
Scriptlet.

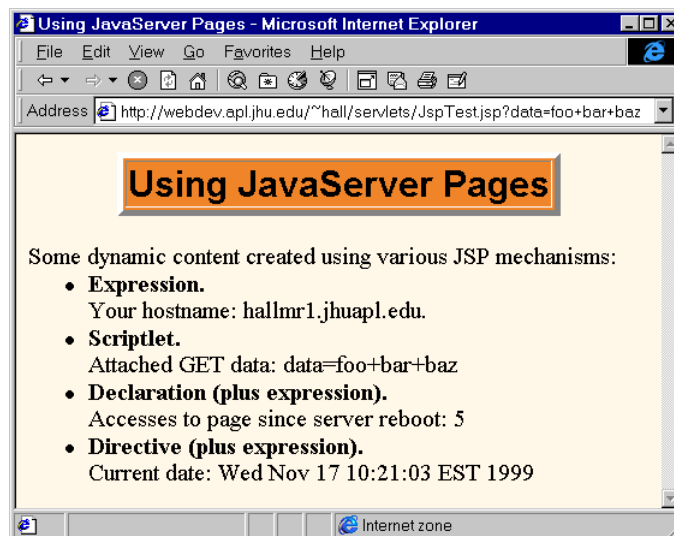
<% out.println("Attached GET data: " +
 request.getQueryString()); %>
Declaration (plus expression).

<%! private int accessCount = 0; %>
Accesses to page since server reboot: <%= ++accessCount %>
Directive (plus expression).

<%@ page import = "java.util.*" %>
Current date: <%= new Date() %>

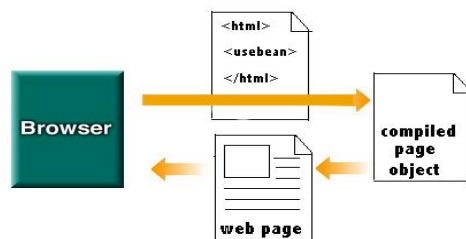
</BODY>
</HTML>
```

## Il risultato

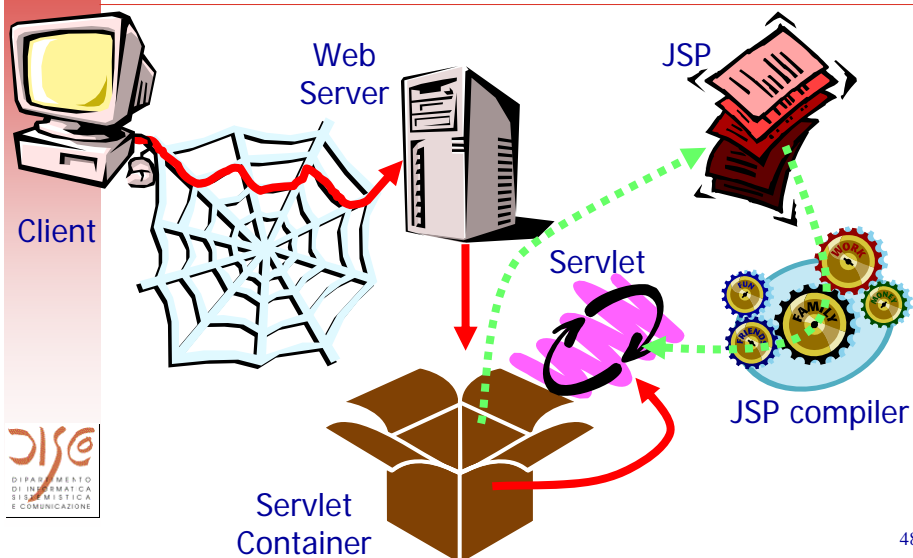


## Come funziona

- In entrambe le modalità, il file JSP viene prima compilato
- la versione compilata viene tenuta in memoria per rendere più veloce una successiva richiesta della pagina



## Ciclo di vita delle applicazioni JSP



## Gli elementi di una JSP

- **Template text**
  - ✓ Le parti statiche della pagina
- **Commenti**
  - <%-- questo e' un commento -->
- **Direttive**
  - <%@ direttiva ... di compilazione %>
- **Azioni**
  - ✓ In XML: <tag attributes> body </tag>
- **Elementi di scripting**
  - ✓ Istruzioni nel linguaggio specificato nelle direttive
  - ✓ Sono di tre tipi: scriptlet, declaration, expression

## Direttive

- **page**
  - ✓ Liste di attributi/valore
  - ✓ Valgono per la pagina in cui sono inseriti
    - <%@ page import="java.util.\*" buffer="16k" %>
    - <%@ page import="java.math.\*, java.util.\*" %>
    - <%@ page session="false" %>
- **include**
  - ✓ Include in compilazione pagine HTML o JSP
    - <%@ include file="copyright.html" %>
- **taglib**
  - ✓ Dichiarazione tag definiti dall'utente implementando opportune classi
    - <%@ taglib uri="TableTagLibrary"
    - prefix="table"%>
    - <table:loop> ... </table:loop>

## Direttive JSP

### ➤ forward

- ✓ determina l'invio della richiesta corrente, eventualmente aggiornata con ulteriori parametri, all'URL indicata

```
<jsp:forward page="login.jsp" %>
 <jsp:param name="username" value="user" />
 <jsp:param name="password" value="pass" />
</jsp:forward>
```

### ➤ include

- ✓ invia dinamicamente la richiesta ad una data URL e ne include il risultato

```
<jsp:include page="login.jsp" %>
```

### ➤ useBean

- ✓ localizza ed istanzia (se necessario) un javaBean nel contesto specificato
- ✓ Il contesto può essere

La pagina, la richiesta, la sessione, l'applicazione

```
<jsp:useBean id="cart" scope="session"
 class="ShoppingCart" />
```

## Elementi di scripting

### ➤ Declaration `<%! declaration [declaration] ...%>`

- ✓ Variabili o metodi usati nella pagina  
`<%! int[] v= new int[10]; %>`
- ✓ Le var valgono per la durata della servlet

### ➤ Expression `<%= expression %>`

- ✓ Una espressione nel linguaggio di scripting che viene valutata e sostituita con il risultato  
`<p>La radice di 2 vale <%= Math.sqrt(2.0) %></p>`

### ➤ Scriptlet `<% codice %>`

- ✓ Frammenti di codice che controllano la generazione della pagina, valutati alla richiesta  

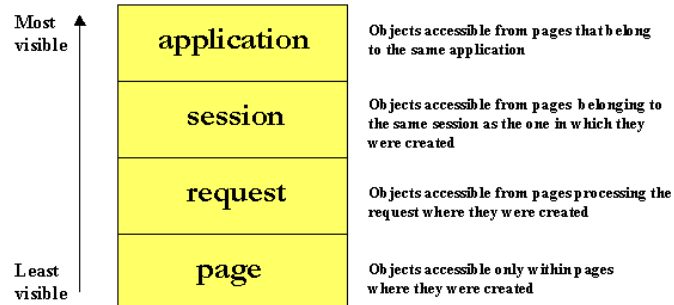
```
<table>
<% for (int i=0; i< v.length; i++) { %>
 <tr><td> <%= v[i] %></td></tr>
<% } %>
</table>
```
- ✓ Le variabili valgono per la singola esecuzione
- ✓ Ciò che viene scritto sullo stream di output sostituisce lo scriptlet

## Elementi di scripting

- Linguaggio di script ha lo scopo di
  - ✓ interagire con oggetti java
  - ✓ gestire le eccezioni java
- Oggetti impliciti
  - ✓ Sono gli elementi delle servlet (sono 9)
    - request
    - response
    - out
    - page
    - pageContext
    - session
    - application
    - config
    - exception

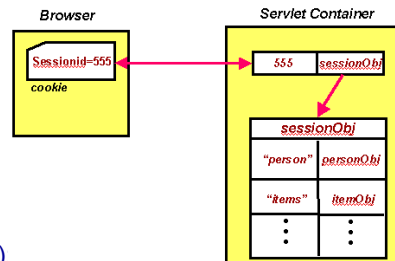
## Oggetti e loro "scope"

- Gli oggetti possono essere creati
  - ✓ implicitamente usando le direttive JSP
  - ✓ esplicitamente con le azioni
  - ✓ direttamente usando uno script (raro)
- Gli oggetti hanno un attributo che ne definisce lo "scope"



## Sessioni

- Accede ad un oggetto HttpSession

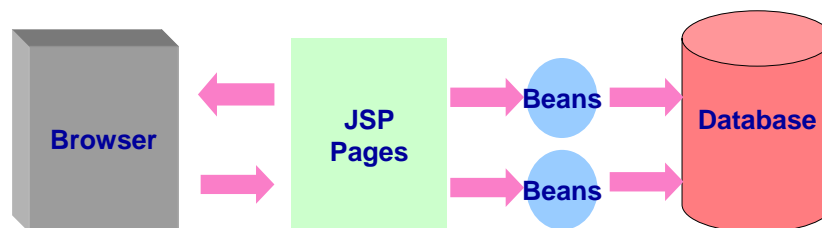


- Esempi

- ✓ Memorizzazione  
`<% Foo foo = new Foo();`
- ✓ Recupero  
`<% Foo myFoo = (Foo) session.getValue("foo"); %>`
- ✓ Esclusione di una pagina dalla sessione  
`<%@ page session="false" %>`

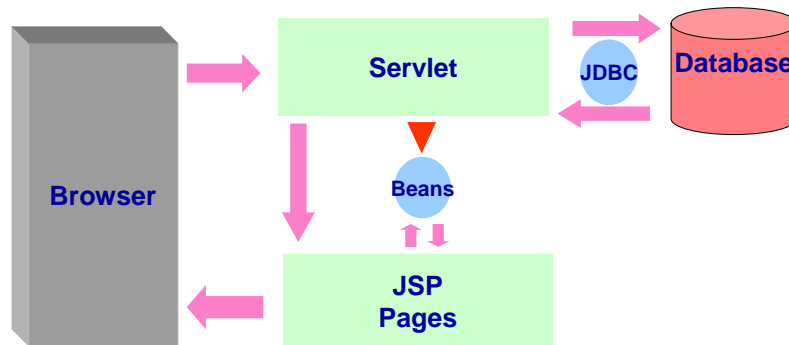
## Esecuzione di JSP

- Il client richiede via HTTP un file .JSP
- Il file .JSP viene interpretato e accede a componenti lato-server (Java Beans, Servlet) che generano contenuti dinamici
- il risultato viene spedito al client sotto forma di pagine HTML



## JSP e Servlet

- la richiesta viene inviata ad un Java Servlet che genera i dati dinamici richiesti dall'utente
- Il Servlet successivamente richiama un file .jsp, che si occupa di formattare in HTML i risultati e inviarli all'utente



## JavaBean

- Un bean è una classe che segue regole precise
  - ✓ Deve avere costruttori senza parametri
  - ✓ Dovrebbe avere campi (property) private
  - ✓ I metodi di accesso ai campi (property) sono set/get setXxx e getXxx/isXxx (xxx = property)

```
class Libro {
 private String titolo;
 private boolean disponibile;
 void setTitolo(String t) ...;
 String getTitolo() ...;
 void setDisponibile(boolean b) ...;
 boolean isDisponibile() ...;
}
```

## JSP e JavaBean

### ➤ Azioni per utilizzare un bean

- ✓ Accedere ad un bean (inizializzazione)

```
<jsp:useBean id="user" class="com.jguru.Person"
scope="session" />
```

```
<jsp:useBean id="user" class="com.jguru.Person"
scope="session">
 <% user.setDate(DateFormat.getDateInstance(
).format(new Date())); //etc.. %>
</jsp:useBean>
```

- ✓ Accedere alle proprietà

```
<jsp:getProperty name="user" property="name" />
<jsp:setProperty name="user" property="name"
value="jGuru" />
<jsp:setProperty name="user" property="name"
value="<%=expression %>" />
```

## Accesso ad un JavaBean

```
<jsp:useBean id="Attore" class="MyThread"
scope="session" type="Thread"/>
```

### ➤ Lo scope determina la vita del bean

- ✓ **page** e' lo scope di default: viene messo in `pageContext` ed acceduto con `getAttribute`
- ✓ **request**: viene messo in `ServletRequest` ed acceduto con `getAttribute`
- ✓ **session** e **application**: se non esiste un bean con lo stesso id, ne viene creato uno nuovo

### ➤ Il **type** permette di assegnargli una superclasse

### ➤ Al posto della classe si puo' usare il nome del bean

- ✓ `beanName="nome"`  
nome e' la classe o un file serializzato

## Counter.jsp

```
1) <%@ page import="CounterBean" %>
2) <jsp:useBean id="session_counter" class="CounterBean"
 scope="session" />
3) <jsp:useBean id="app_counter" class="CounterBean"
 scope="application" />
4) <% session_counter.increaseCount();
5) synchronized(page) { app_counter.increaseCount(); }
6) %>
7) <h3>
8) Number of accesses within this session:
9) <jsp:getProperty name="session_counter" property="count" />
10) </h3>
11) <p>
12) <h3>
13) Total number of accesses:
14) <% synchronized(page) { %>
15) <jsp:getProperty name="app_counter" property="count" />
16) <% } %>
17) </h3>
```

## CounterBean.java

```
public class CounterBean {
 //declare a integer for the counter
 int count;

 public int getCount() {
 //return count
 return count;
 }

 public void increaseCount() {
 //increment count;
 count++;
 }
}
```