Programmazione WEB

Lezione del 5 Marzo 2020

Docente: Novella Bartolini
Ricevimento: Mercoledì ore 12:30-14:30
Via Salaria 113, terzo piano, stanza 309
Email: bartolini@di.uniroma1.it

Interazione Client-Server WEB

- Sempre basata sul protocollo HTTP indipendentemente dalla soluzione adottata dal lato del server
- L'URL oggetto della richiesta è usato per selezionare la risorsa lato server che si vuole usare

Protocollo HTTP

- Protocollo stateless
  - Ad ogni richiesta del client segue una risposta del server. Nessuna correlazione tra richieste successive.
- Quando un client invia una richiesta al server, specifica un comando
- La prima linea della richiesta contiene il nome del comando, un URL e la versione del protocollo in uso:

```
GET /main.html HTTP/1.0
```

Protocollo HTTP (segue)

- Alla richiesta vengono appese informazioni opzionali
  - versione del browser,
  - i tipi di file che possono essere elaborati…

Protocollo HTTP (segue)

- Il server elabora la richiesta e spedisce una risposta, specificando la versione del protocollo e uno status code (header della risposta):

```
HTTP/1.1 200 OK
```

- Altre informazioni inviate nell'header della risposta
  - Server software
  - Content-type della risposta

Protocollo HTTP: GET & POST

- GET e POST sono i comandi HTTP utilizzati più frequentemente
- Progettati inizialmente per scopi diversi
  - GETting information
  - POSTing information
- GET: informazioni utili per formulare la richiesta appese all'URL
- POST: informazioni incluse nel corpo della richiesta

Protocollo HTTP: GET

- I parametri della richiesta sono visibili
- La lunghezza della query string è limitata dal browser
  - Molti browser non consentono l'uso di query string di più di 240 caratteri
- La richiesta può essere inserita nei bookmark e ripetuta quante volte si vuole

Protocollo HTTP: POST

- I parametri della richiesta non sono visibili all'utente
- La quantità di dati che si possono inviare è illimitata (anche megabytes)
- Le richieste di POST non possono essere inserite nei bookmark, né spedite via email o ricaricate.

Protocollo HTTP: GET e POST

- La distinzione funzionale con cui i metodi erano stati progettati si è persa MA ricorda
- GET
  - parametri visibili
  - lista breve
  - inseribile nei bookmark e ripetibile
- POST:
  - parametri nascosti
  - lunghezza illimitata
  - non inseribile nei bookmark e non ripetibile
- Non usare richieste di GET per gestire ordini o aggiornamenti di un database

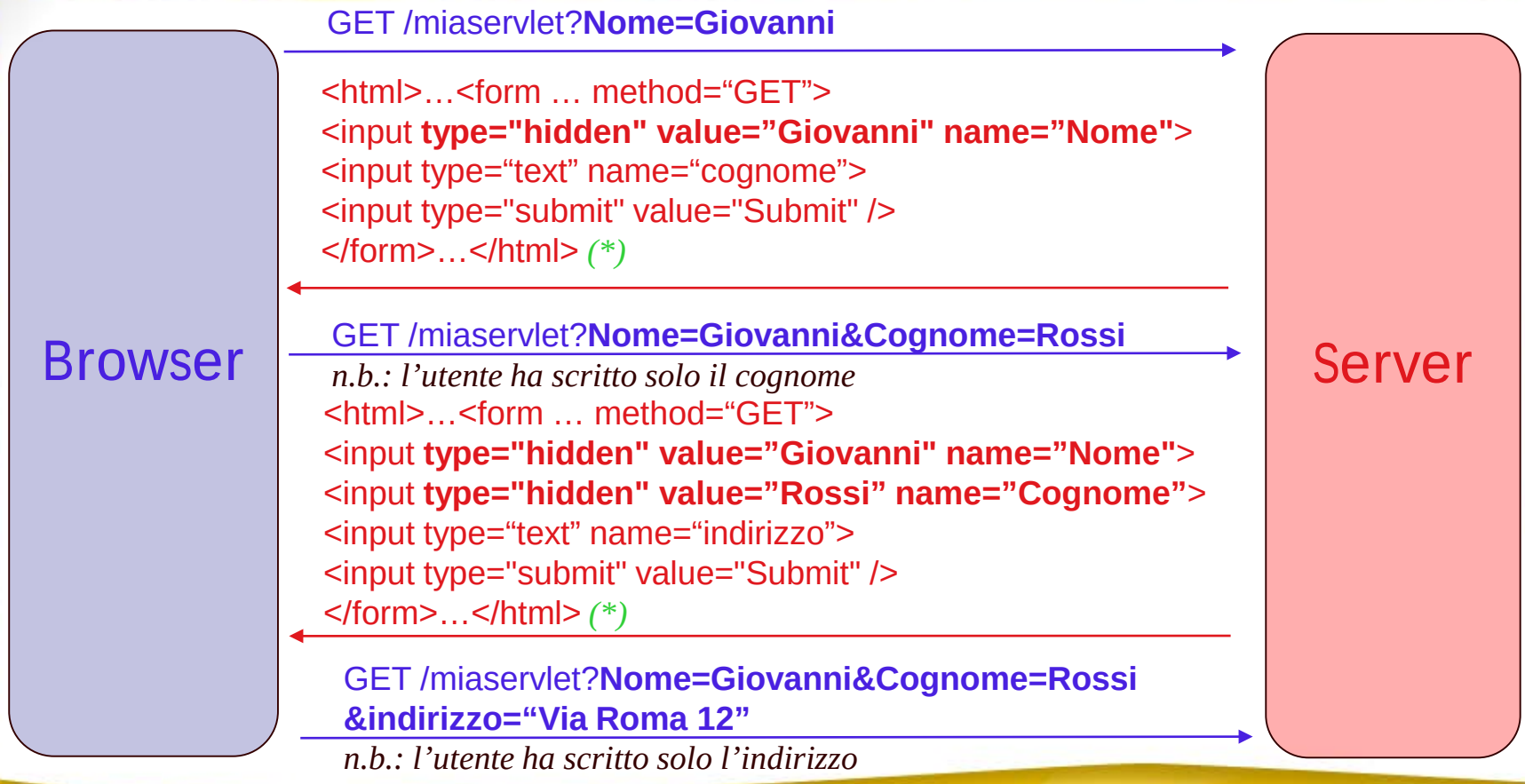
Caratteristiche del protocollo HTTP

E' il protocollo per il trasferimento di iper-testi (HyperText Transfer Protocol)

- Corrisponde al livello applicativo (stack iso/osi)
  - Presuppone un protocollo (TCP) orientato alla connessione
- Meccanismo di Richiesta/Risposta (Client/Server)
- E' possibile un trasferimento bi-direzionale di informazioni
  - il client può inviare informazioni a un server tramite un form, il server invia (di solito) pagine web al client
- Basato sul meccanismo di naming degli URI
- **Senza stato**
  - **Ogni richiesta HTTP è autonoma, il server non tiene una cronologia delle richieste**

Gestione della sessione tramite parametri hidden

Gestione della sessione tramite parametri hidden



(*) Il testo della pagina html viene generato dinamicamente dalla servlet in funzione dei parametri ricevuti nella richiesta, sia che fossero hidden sia che fossero visibili nel form

Sviluppo di applicazioni di rete

- Paradigma client-server
- Implicazioni del paradigma client-server nel funzionamento dei sistemi web
  - Funzionamento di un web server
  - Generazione di pagine dinamiche con tecnologie lato client e lato server

Tecnologie lato client o lato server?

lato client all'interno di Applet, o attraverso metodi di scripting dal lato client (JavaScript), o in applicazioni stand-alone.

Richiedono il supporto del client

lato server associato a Servlet, agli Enterprise JavaBean, agli Agenti, alle API ed ai servizi di Transaction Management, agli Application Server ed ai protocolli di programmazione distribuita (es. CORBA)

Non richiedono alcun supporto da parte del client

Perché soluzioni lato server

- Soluzioni lato client
  - problemi di prestazioni e di portabilità
- Soluzioni lato server
  - accesso a informazioni che sono disponibili esclusivamente dal lato del server (es. database etc.)
  - minimi requisiti in termini di capacità di calcolo e storage dal lato del client (è il server a fare il grosso del lavoro)
  - il client non deve avere altro che il browser per interpretare le pagine html

Servlet

- Programma applicativo che viene eseguito da un server
  - Accoglie ed elabora richieste provenienti dal client (attraverso comandi http: POST o GET, ma anche attraverso altri protocolli)
  - Produce una risposta HTTP contenente codice HTML generato dinamicamente
- Molto diffuse per applicazioni che fanno uso di database
  - Thin clients (client molto semplici che richiedono supporto minimo)
  - Meccanismo request/response

Servlet: meccanismi implementati

- Identificazione della sessione, ovvero di una serie di richieste provenienti da un'unica coppia utente/browser
- Accesso alle funzioni native di autenticazione fornite dal sistema operativo

  e a seconda dell'engine altri come…
- Clustering delle sessioni (per load balancing)
- Supporto profilazione persistente di utenti tramite JDBC
- Connettività a database

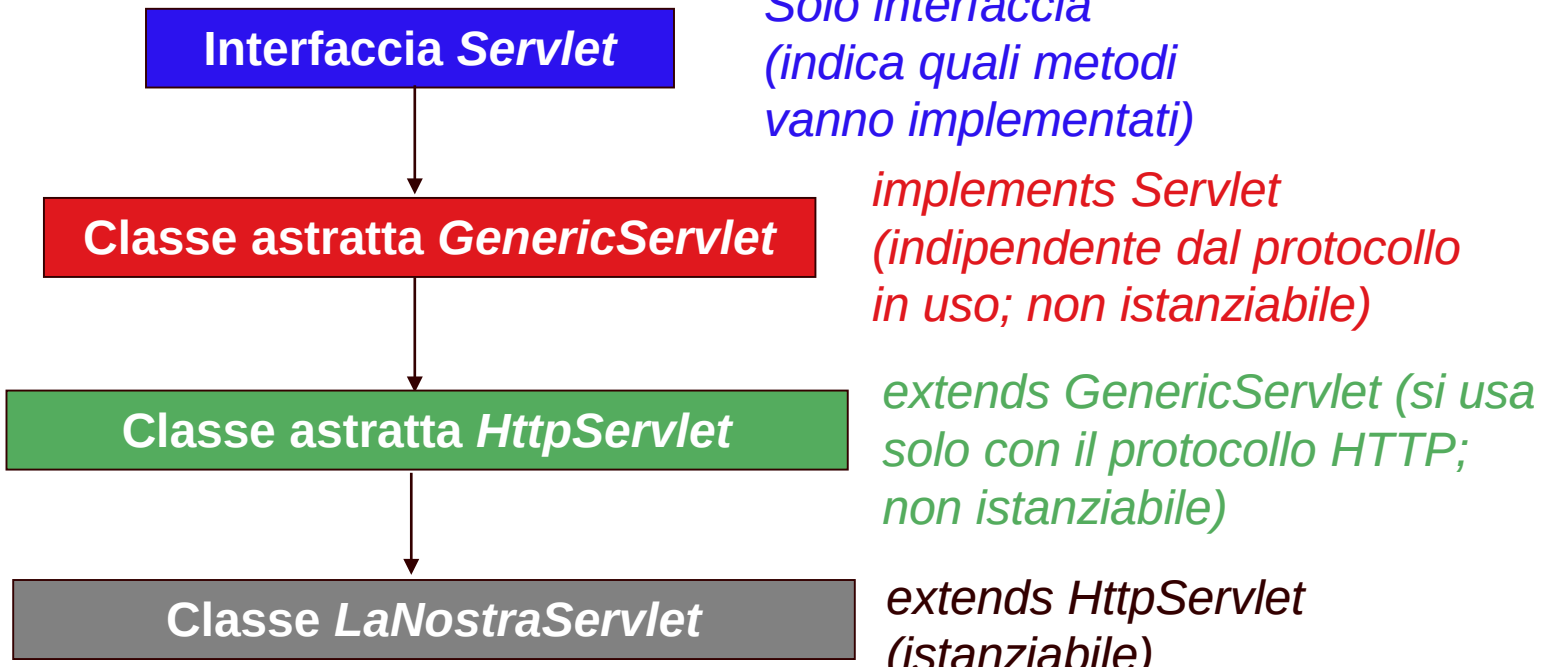
Servlet e Servlet Container

- Una servlet è una classe Java (che implementa l'interfaccia `Servlet`).
- Un servlet container può ospitare più servlet (con relativi alias).
- Quando una servlet viene invocata per la prima volta, il servlet engine genera un thread Java che inizializza l'oggetto Servlet.
  - Questo *persiste* per tutta la durata del processo relativo al servlet container (salvo esplicita de-allocazione).
- Ogni servlet è un thread all'interno del Servlet Container (vs CGI dove viene eseguito un processo esterno)

Possibili usi di una Servlet

- Supportare richieste multiple in modo concorrente, come per esempio conferenze on-line, servizi di comunicazione
- Aggiornare, eliminare o consultare dati contenuti in un DB remoto tramite protocollo TCP/IP
- Applicazioni basate sull'autenticazione degli utenti, gestione di sessioni di navigazione con creazione di oggetti persistenti
- Ottimizzazione delle prestazioni tramite redirezione di richieste ad altre Servlet in altri server, allo scopo di bilanciare il carico di lavoro

Servlet API



Solo interfaccia (indica quali metodi vanno implementati)

implements Servlet (indipendente dal protocollo in uso; non istanziabile)

extends GenericServlet (si usa solo con il protocollo HTTP; non istanziabile)

extends HttpServlet (istanziabile)

Servlet

- Le servlet possono essere utilizzate qualunque sia il servizio espletato dal server, ovvero qualunque sia il protocollo di interazione client/server: es HTTP, FTP
- Nella sua forma più generale, una servlet è un'estensione di una classe javax.servlet.GenericServlet che implementa l'interfaccia javax.servlet.Servlet
- Le servlet usate nel web sono estensioni della classe javax.servlet.http.HttpServlet che implementa l'interfaccia javax.servlet.Servlet

Interfaccia `Servlet`

- Interfaccia `Servlet` (package *`javax.servlet`*)
  - Tutte le servlet devono implementare questa interfaccia
  - Tutti i metodi dell'interfaccia `Servlet` vengono invocati dal servlet container secondo un prefissato *ciclo di vita*
    - (vi ricordate come funziona con le applet? Init-start-paint-stop-destroy vengono sempre chiamati in sequenza dal browser per eseguire una applet)

Ciclo di vita di una servlet

- Caricamento della servlet in memoria
- Il container delle servlet invoca il metodo `init`
  - Solo in questo momento la servlet è in grado di rispondere alla prima richiesta
  - Inizializzazione delle variabili globali
  - Invocato una sola volta
- Il metodo `service` gestisce le richieste
  - Riceve la richiesta
  - Elabora la richiesta
  - Confeziona un oggetto risposta
  - Invocato ad ogni richiesta del client
- Il metodo `destroy` rilascia le risorse allocate dalla servlet quando il container la termina

Classi astratte per definire le Servlet

- Esistono due classi astratte che implementano l'interfaccia *Servlet*
  - **`GenericServlet` (package `javax.servlet`)**
  - **`HttpServlet` (package `javax.servlet.http`)**
    (quest'ultima implementa l'interfaccia servlet indirettamente, estendendo GenericServlet)
- Queste classi forniscono l'implementazione di default di tutti i metodi dell'interfaccia *Servlet*
- Il metodo chiave è *service*:

  riceve gli oggetti ServletRequest e ServletResponse che forniscono accesso agli stream di i/o permettendo la ricezione e l'invio di informazioni al client

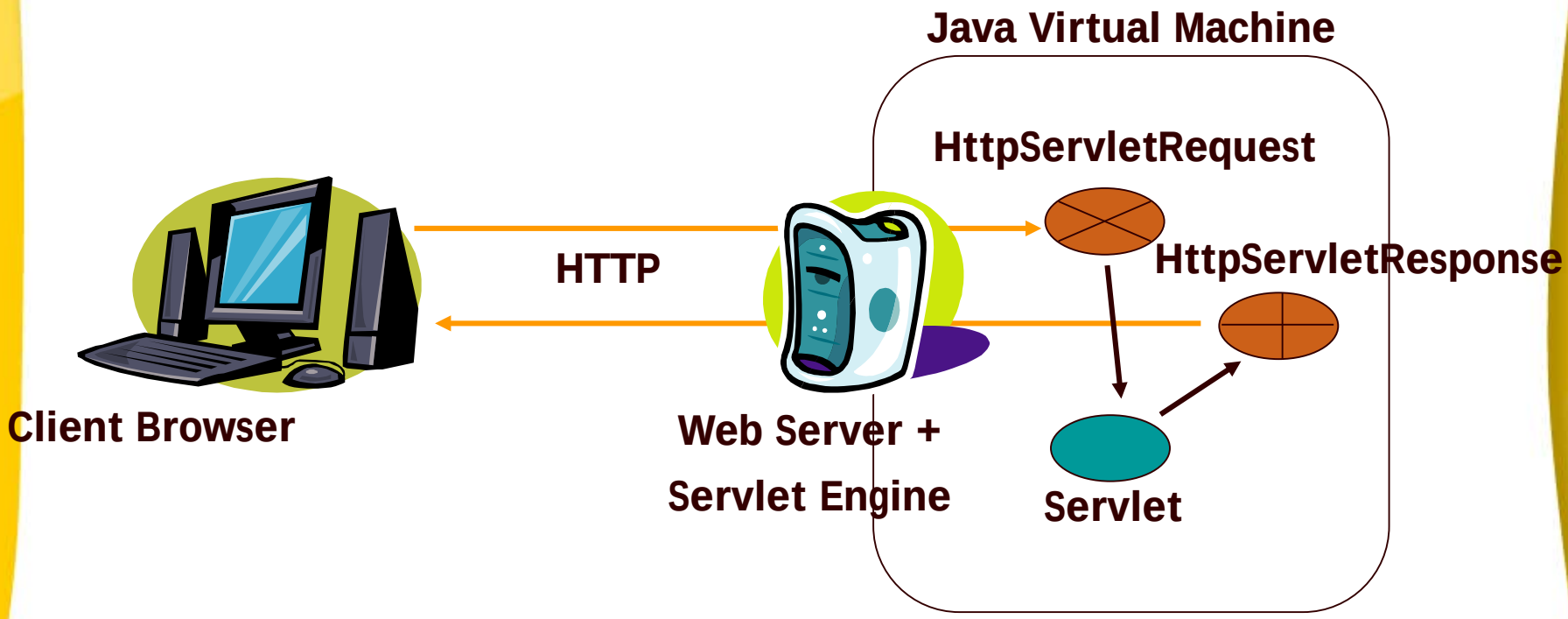
Metodi dell'interfaccia `Servlet`

Method	Description
`void init( ServletConfig config )`	
	The servlet container calls this method once during a servlet's execution cycle to initialize the servlet. The `ServletConfig` argument is supplied by the servlet container that executes the servlet.
`ServletConfig getServletConfig()`	
	This method returns a reference to an object that implements interface `ServletConfig`. This object provides access to the servlet's configuration information such as servlet initialization parameters and the servlet's `ServletContext`, which provides the servlet with access to its environment (i.e., the servlet container in which the servlet executes).
`String getServletInfo()`	
	This method is defined by a servlet programmer to return a string containing servlet information such as the servlet's author and version.
`void service( ServletRequest request, ServletResponse response )`	
	The servlet container calls this method to respond to a client request to the servlet.
`void destroy()`	
	This "cleanup" method is called when a servlet is terminated by its servlet container. Resources used by the servlet, such as an open file or an open database connection, should be deallocated here.

Funzionamento delle servlet HTTP

- Il server web riceve dal browser del client una richiesta HTTP GET o POST
- Il server web direziona la richiesta HTTP al motore servlet o servlet engine (pr. èngine) o servlet container
- Se la servlet non è ancora in memoria viene caricata e inizializzata dal servlet engine (esecuzione del metodo init)
- Il servlet engine incapsula la richiesta HTTP in un oggetto di classe **HttpServletRequest** e lo passa al metodo doPost o doGet della servlet
- La servlet risponde scrivendo il codice HTML nell'oggetto di classe **HttpServletResponse** che viene rimandato al web server e poi riconsegnato al client via HTTP

Servlet e Web Server



La classe `HttpServlet`

- Sovrascrive il metodo `service` della classe `GenericServlet`
- Prevede i metodi per rispondere alle richieste HTTP
  - `GET`
  - `POST (ma anche HEAD, PUT, OPTIONS ecc.)`
- Il metodo `service()` invoca i metodi `doGet` e `doPost`
  - Il metodo `doGet()` risponde alle richieste `GET`
  - Il metodo `doPost()` risponde alle richieste `POST`
    - Ricevono come parametri gli oggetti `HttpServletRequest` e `HttpServletResponse`
- `Non implementare il metodo service se si usa questo tipo di servlet`

Altri metodi della classe `HttpServlet`

Method	Description
`doDelete`	Called in response to an HTTP *`delete`* request. Such a request is normally used to delete a file from a server. This may not be available on some servers, because of its inherent security risks (e.g., the client could delete a file that is critical to the execution of the server or an application).
`doHead`	Called in response to an HTTP *`head`* request. Such a request is normally used when the client only wants the headers of a response, such as the content type and content length of the response.
`doOptions`	Called in response to an HTTP *`options`* request. This returns information to the client indicating the HTTP options supported by the server, such as the version of HTTP (1.0 or 1.1) and the request methods the server supports.
`doPut`	Called in response to an HTTP *`put`* request. Such a request is normally used to store a file on the server. This may not be available on some servers, because of its inherent security risks (e.g., the client could place an executable application on the server, which, if executed, could damage the server—perhaps by deleting critical files or occupying resources).
`doTrace`	Called in response to an HTTP *`trace`* request. Such a request is normally used for debugging. The implementation of this method automatically returns an HTML document to the client containing the request header information (data sent by the browser as part of the request).

Interfaccia `HttpServletRequest`

- Il servlet engine che esegue la servlet
  - Crea un oggetto **`HttpServletRequest`**
  - Lo passa al metodo **`service`** della servlet http
- L'oggetto **`HttpServletRequest`** contiene la richiesta del client
- Esistono molti metodi per estrarre informazioni dalla richiesta del client. Ne vediamo alcuni →

Interfaccia `HttpServletRequest` (Cont.)

Method	Description
`String getParameter( String name )`	
	Obtains the value of a parameter sent to the servlet as part of a `get` or `post` request. The name argument represents the parameter name.
`Enumeration getParameterNames()`	
	Returns the names of all the parameters sent to the servlet as part of a `post` request.
`String[] getParameterValues( String name )`	
	For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter.
`Cookie[] getCookies()`	
	Returns an array of `Cookie` objects stored on the client by the server. `Cookie` objects can be used to uniquely identify clients to the servlet.
`HttpSession getSession( boolean create )`	
	Returns an `HttpSession` object associated with the client's current browsing session. This method can create an `HttpSession` object (`true` argument) if one does not already exist for the client. `HttpSession` objects are used in similar ways to `Cookie`s for uniquely identifying clients.

Interfaccia `HttpServletResponse`

- Il servlet engine
  - Crea un oggetto `HttpServletResponse`
  - Lo passa al metodo `service` della servlet (attraverso i metodi doGet e doPost)
- Ogni chiamata ai metodi `doGet` e `doPost` riceve un oggetto che implementa l'interfaccia `HttpServletResponse`
- Esistono molti metodi che consentono la scrittura della risposta da inviare al client. Ne vediamo alcuni →

Interfaccia `HttpServletResponse`

Method	Description
void addCookie(Cookie cookie)	
	Used to add a Cookie to the header of the response to the client. The CookieÕs maximum age and whether Cookies are enabled on the client determine if Cookies are stored on the client.
ServletOutputStream getOutputStream()	
	Obtains a byte-based output stream for sending binary data to the client.
PrintWriter getWriter()	
	Obtains a character-based output stream for sending text data to the client.
void setContentType(String type)	
	Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type "text/html" indicates that the response is an HTML document, so the browser displays the HTML page.

Servlet container

Architetture per l'esecuzione di servlet

- Servlet container (o servlet engine)
  - E' il server che esegue una servlet
- Servlet e JSP sono supportate direttamente o attraverso plugin dalla maggior parte dei Web servers e application server
  - Apache TomCat
  - Sun ONE Application Server
  - Microsoft's Internet Information Server (IIS)
  - Apache HTTP Server + modulo JServ
  - BEA's WebLogic Application Server
  - IBM's WebSphere Application Server
  - World Wide Web Consortium's Jigsaw Web Server

Configurare il server Apache Tomcat (1)

1. Installare j2sdk
2. Definire la variabile PATH=c:\j2sdk---\bin (per poter trovare il compilatore)
3. Definire la variabile JAVAHOME=c:\j2sdk--- (perché tomcat trovi la jvm)
4. Installare il server TomCat

http://tomcat.apache.org/download-70.cgi

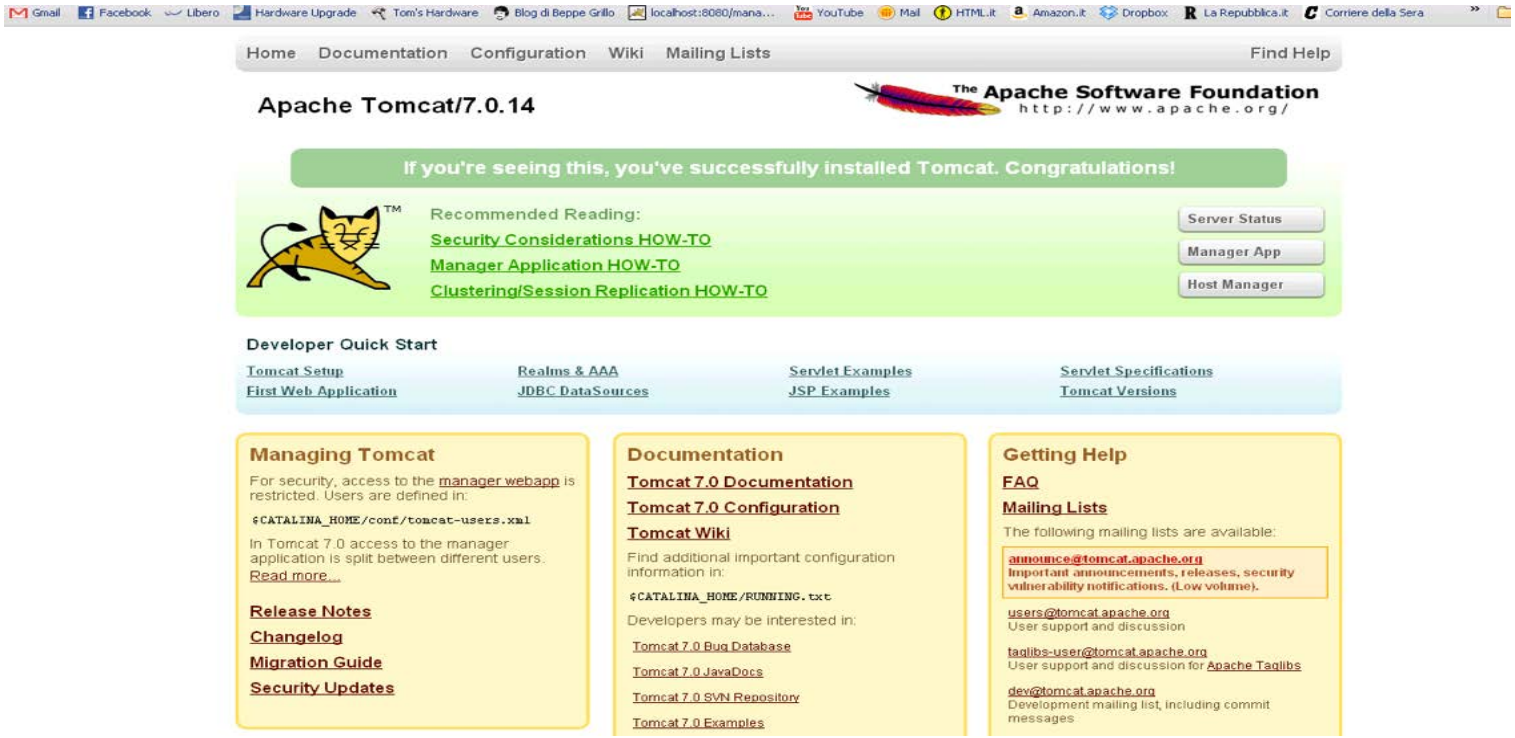
Configurare il server Apache Tomcat (2)

5. Definire la variabile CATALINA_HOME=c:\Programmi\Apache…\Tomcat---\

6. Definire la variabile CLASSPATH=c:\Programmi\Apache…\Tomcat---\common\lib\servlet-api.jar;c:\Programmi\Apache…\Tomcat---\common\lib\jsp-api.jar

7. TEST: Avviare il server Tomcat (startup), invocare il server da un browser all'indirizzo http://localhost:8080/

Configurare il server Apache Tomcat



Se Tomcat è installato correttamente dovreste vedere questa pagina all'indirizzo http://127.0.0.1:8080/

```
1
2  // A simple servlet to process get requests.
3
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6  import java.io.*;
7
8  public class ServletDiSaluto extends HttpServlet {
9
10   // process "get" requests from clients
11   protected void doGet( HttpServletRequest request,
12     HttpServletResponse response )
13       throws ServletException, IOException
14   {
15     response.setContentType( "text/html" );
16     PrintWriter out = response.getWriter();
17
18     // send XHTML page to client
19
20     // start XHTML document
21     out.println( "<?xml version = \"1.0\"?>" );
22
23     out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
24       "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
25       "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
```

Importa i package `javax.servlet` e `javax.servlet.http`.

Estende `HttpServlet` in modo che gestisca le richieste HTTP `get` e `post`.

Sovrascrive il metodo `doGet` per fornire funzionalità personalizzate.

Utilizza l'oggetto `response` e il relativo metodo `setContentType` per specificare il tipo di contenuto dei dati che devono essere spediti in risposta al client.

Utilizza il metodo `getWriter` dell'oggetto `response` per ottenere un riferimento all'oggetto `PrintWriter` che abilita la servlet a spedire dati al client.

Crea il documento XHTML scrivendo stringhe attraverso il metodo `println` dell'oggetto `out`.

ServletDiSaluto.java

```
26
27        out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
28
29        // head section of document
30        out.println( "<head>" );
31        out.println( "<title>A Simple Servlet Example</title>" );
32        out.println( "</head>" );
33
34        // body section of document
35        out.println( "<body>" );
36        out.println( "<h1>Welcome to Servlets!</h1>" );
37        out.println( "</body>" );
38
39        // end XHTML document
40        out.println( "</html>" );
41        out.close();  // close stream to complete the page
42     }
43  }
```

Chiude l'output stream, esegue il flush del buffer di output e spedisce le informazioni al client.

ServletDiSaluto.java
(2 of 2)

File index.html

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!Ń File index.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Handling an HTTP Get Request</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/Alias ServletDiSaluto" method = "get">
14
15     <p><label>Click the button to invoke the servlet
16       <input type = "submit" value = "Get HTML Document" />
17     </label></p>
18
19   </form>
20 </body>
21 </html>
```

Struttura della directory di una web application

- Context root – top level directory di una applicazione web

  Si crea una sottodirectory nella directory webapps di Tomcat

Struttura della directory di una web application

Directory	Description
context root (nome che volete)	E' la directory radice della vostra applicazione web. Il nome è scelto dallo sviluppatore dell'applicazione (voi). Contiene tutti i file dell'applicazione, eventualmente organizzati in sottocartelle (file JSP, pagine HTML, immagini incorporate nelle pagine, possono risiedere dove volete, classi delle servlet e altre classi di supporto devono essere organizzate in cartelle e package. Si possono inserire sottocartelle nella context root per organizzarne meglio i contenuti. Negli esempi che seguono la context root si chiama **`directorydisaluto`**.
`WEB-INF`	Questa cartella contiene un file detto *deployment descriptor* (***`web.xml`***).
`WEB-INF/classes`	Questa cartella contiene le classi delle servlet e altre classi di supporto usate dalla web application. Se le classi fanno parte di un package, la struttura completa del package deve cominciare da questa cartella.
`WEB-INF/lib`	Questa cartella contiene archivi (JAR). I file JAR files possono contenere I file delle classi servlet e altre classi di supporto che sono utilizzate in una web application.

Deployment descriptor (web.xml)

- Dopo avere configurato l'albero delle directory, dobbiamo configurare l'applicazione web in modo che possa gestire le richieste

→ Per la configurazione di molti aspetti funzionali della web application si usa un file web.xml che chiamiamo

deployment descriptor

In particolare, nella gestione delle servlet, il web.xml specifica diversi parametri come:

- il nome della classe che definisce la servlet,
- i percorsi che causano l'invocazione della servlet

```
1  <!DOCTYPE web-app PUBLIC
2    "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4
5  <web-app>
6
7    <!-- General description of your W
8    <display-name>
9     NomeDisplayDellaWebApplication
10      Viene visualizzato dall'admin
11   </display-name>
12
13   <description>
14      Si tratta di un'applcazione dove facciamo
15      Esempi di servlet.
16   </description>
17
18   <!-- Servlet definitions -->
19    <servlet>

20      <servlet-name>Paperino</servlet-nam
21
22      <description>
23        Una servlet che gestisce le richieste di GET
24      </description>
25
26      <servlet-class>
27        ServletDiSaluto
28      </servlet-class>
29   </servlet>
30
```

L'elemento <**web-app**> definisce la configurazione di tutte le servlet della applicazione web

L'elemento <**display-name**> specifica un nome che viene utilizzato dall'amministratore del server (e visualizzato dal pannello del manager)

L'elemento <**description**> specifica una descrizione della applicazione che viene visualizzata nel pannello del manager

L'elemento <**servlet**> descrive una specifica servlet: se ci sono più servlet ci saranno più elementi <servlet>

L'elemento <**servlet-name**> definisce un nome per la servlet all'interno di questo file

L'elemento <**description**> fornisce una descrizione di questa servlet specifica

L'elemento <**servlet-class**> specifica il nome completo della classe servlet compilata

```
31    <!-- Servlet mappings -->
32    <servlet-mapping>
33       <servlet-name>Paperino</servlet-name>
34       <url-pattern>/AliasServletDiSaluto</url-pattern>
35    </servlet-mapping>
36
37 </web-app>
```

L'elemento <**servlet-mapping**> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<**servlet-name**>) e l'<**url-pattern**> con cui la servlet potrà essere invocata

n.b.: non si specifica la context root nell'url-pattern!

Ricordiamoci che all'interno del server la tecnologia è java, mentre il colloquio con il client avviene nel protocollo HTTP.

L'associazione tra l'URL usata nelle richieste HTTP e la classe java che deve essere eseguita dal server è lo scopo principale del deployment descriptor.

Tale associazione è creata nel file web.xml attraverso l'elemento <servlet-name> che viene associato prima con il nome della classe (<servlet-class>) e poi con l'URL (<url-pattern>).

n.b. se la servlet fosse stata parte di un package, avremmo dovuto includere nella directory **classes** l'intera struttura del package

```
31    <!-- Servlet mappings -->
32    <servlet-mapping>
33       <servlet-name>Paperino</servlet-name>
34       <url-pattern>/A/B/topolino</url-pattern>
35    </servlet-mapping>
36
37 </web-app>
```

L'elemento <**servlet-mapping**> specifica l'associazione tra il nome dato alla servlet nell'interno del file web.xml (<**servlet-name**>) e l'<**url-pattern**> con cui la servlet potrà essere invocata

Notare che /A e /B non corrispondono a nessuna cartella sul server!

Servlet annotations (1/3)

- Il package javax.servlet.annotation consente la configurazione di una servlet tramite annotazioni che possono sostituire il file web.xml
- Possiamo configurare l'URL pattern di una servlet utilizzando l'annotazione @WebServlet direttamente nel testo del file sorgente.
- Ad esempio, questa annotazione configura il percorso /AliasServletDiSaluto per la servlet precedente

```
@WebServlet("/AliasServletDiSaluto")
public class ServletDiSaluto extends HttpServlet {
```

Servlet annotations (2/3)

```
@WebServlet("/AliasServletDiSaluto")
public class ServletDiSaluto extends HttpServlet {
```

- Questa annotazione indica che l'URL pattern /AliasServletDiSaluto segue la context root. Quindi, compilando il file con questa annotazione, la servlet sarà accessibile attraverso l'URL:

  http://host:port/DirectoryDiSaluto/AliasServletDiSaluto

- Per utilizzare la servlet invocando la sola context root, specificare "/" come URL pattern.

Servlet annotations (3/3)

- Il package javax.servlet.annotation consente la configurazione di una servlet tramite annotazioni che possono sostituire il file web.xml
- La scelta tra usare web.xml o annotazioni è soggettiva, non si riflette in un miglioramento delle prestazioni.
  - Con il web.xml il codice rimane pulito e modulare
  - Il web.xml serve per configurare molti altri aspetti di una applicazione, oltre al mapping delle servlet
  - Con il web.xml se vogliamo cambiare la configurazione delle nostre servlet non siamo costretti a ricompilarne il codice

Uso di percorsi relativi in una Web Application (1)

- Invocazione della servlet attraverso un **percorso relativo al server**:

  "/DirectoryDiSaluto/AliasServletDiSaluto"

  - **`/DirectoryDiSaluto`** specifica la context root
  - **`/AliasServletDiSaluto`** specifica l'URL pattern
  - Si tratta di un percorso relativo al server riferito alla radice del server (dir. webapps), comincia con il carattere "/"
  - Questo percorso va bene qualunque sia la posizione del file chiamante all'interno del server (anche al di fuori della context root, da un'altra applicazione purchè all'interno dello stesso server)

```
<form action = "/DirectoryDiSaluto/AliasServletDiSaluto" method = "get">
```

Uso di percorsi relativi in una web application (2)

- Invocazione della servlet attraverso un **percorso relativo al file chiamante**:

  "percorso_x/AliasServletDiSaluto"

  - percorso_x specifica il percorso dalla cartella contenente il file chiamante alla context root dell'applicazione
  - Si tratta di un percorso relativo al file chiamante riferito alla posizione del file in cui è specificato l'url della servlet che si vuole invocare.
  - Il percorso relativo comincia senza il carattere "/"
  - Questo percorso va bene solo per la posizione del file chiamante e non può essere utilizzato in altre applicazioni, nemmeno se girano sullo stesso server

Uso dei percorsi relativi in una web application (3)

WelcomeServlet Web application directory and file structure

```
+ DirectoryDiSaluto
    - index.html
    + htmls
        - HtmlDiSaluto.html
    + WEB-INF
        - web.xml
        + classes
          -  ServletDiSaluto.class
```

Nel file web.xml l'url pattern indicato è
<url-pattern>/AliasServletDiSaluto</url-pattern>

Nel file index.html la servlet viene invocata nel seguente modo:

```
<form action = "AliasServletDiSaluto" method = "get">
```

Nel file HtmlDiSaluto.html la servlet viene invocata nel seguente modo:

```
<form action = "../AliasServletDiSaluto" method = "get">
```

Gestione di una richiesta HTTP `get` Contenente Dati

- Servlet **`WelcomeServlet2`**
  - Rispondere ad una richiesta di tipo **`get`** contenente dati

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Processing get requests with data</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/welcome2" method = "get">
14
15     <p><label>
16       Type your first name and press the Submit button
17       <br /><input type = "text" name = "firstname" />
18       <input type = "submit" value = "Submit" />
19     </p></label>
20
21   </form>
22 </body>
23 </html>
```

Get the first name from the user.

```
2  // Processing HTTP get requests containing data.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet2 extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13      HttpServletResponse response )
14        throws ServletException, IOException
15    {
16      String firstName = request.getParameter( "firstname" );
17
18      response.setContentType( "text/html" );
19      PrintWriter out = response.getWriter();
20
21      // send XHTML document to client
22
23      // start XHTML document
24      out.println( "<?xml version = \"1.0\"?>" );
25
26      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
27        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
28        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
29
30      out.println(
31        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
32
```

Il metodo **getParameter** dell'oggetto **request** riceve come argomento il nome del parametro e restituisce il corrispondente vaolre sotto forma di una stringa (tipo **String**)

```
33      // head section of document
34      out.println( "<head>" );
35      out.println(
36        "<title>Processing get requests with data</title>" );
37      out.println( "</head>" );
38
39      // body section of document
40      out.println( "<body>" );
41      out.println( "<h1>Hello " + firstName + ",<br />" );
42      out.println( "Welcome to Servlets!</h1>" );
43      out.println( "</body>" );
44
45      // end XHTML document
46      out.println( "</html>" );
47      out.close();  // close stream to complete the page
48   }
49 }
```

La stringa ottenuta alla linea 16 viene usata per costruire la risposta al client.

Modifica del file web.xml per includere la nuova servlet

Descriptor element	Value
servlet *element*	
servlet-name	**Welcome2**
description	**Handling HTTP get requests with data.**
servlet-class	**WelcomeServlet2**
servlet-mapping *element*	
servlet-name	**Welcome2**
url-pattern	**/Welcome2**

Modifica del file web.xml per includere la nuova servlet

- Si aggiungono le seguenti direttive:

```
<servlet>
   <servlet-name>Welcome2</servlet-name>
   <display-name>
     NomeDisplayDiSalutoPersonalizzato</display-name>
   <description>Facciamo un test con personalizzazione</description>
   <servlet-class>WelcomeServlet2</servlet-class>
 </servlet>

<servlet-mapping>
   <servlet-name>Welcome2</servlet-name>
   <url-pattern>/Welcome2</url-pattern>
 </servlet-mapping>
```


Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/servlets/WelcomeServlet2.html
Go
Type your first name and press the Submit button
Paul
Submit
Done
Processing get requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/welcome2?firstname=Paul
Go
Hello Paul,
Welcome to Servlets!
Done
Local intranet

Servlet che gestisce richieste di POST

- HTTP POST request
  - Spedire dati da un form HTML ad un handler di form localizzato sul server
  - Ricordare che i browser non fanno caching delle risposte a richieste di tipo POST
- Servlet WelcomeServlet3
  - Risponde ad una richiesta di tipo **POST** contenente dati

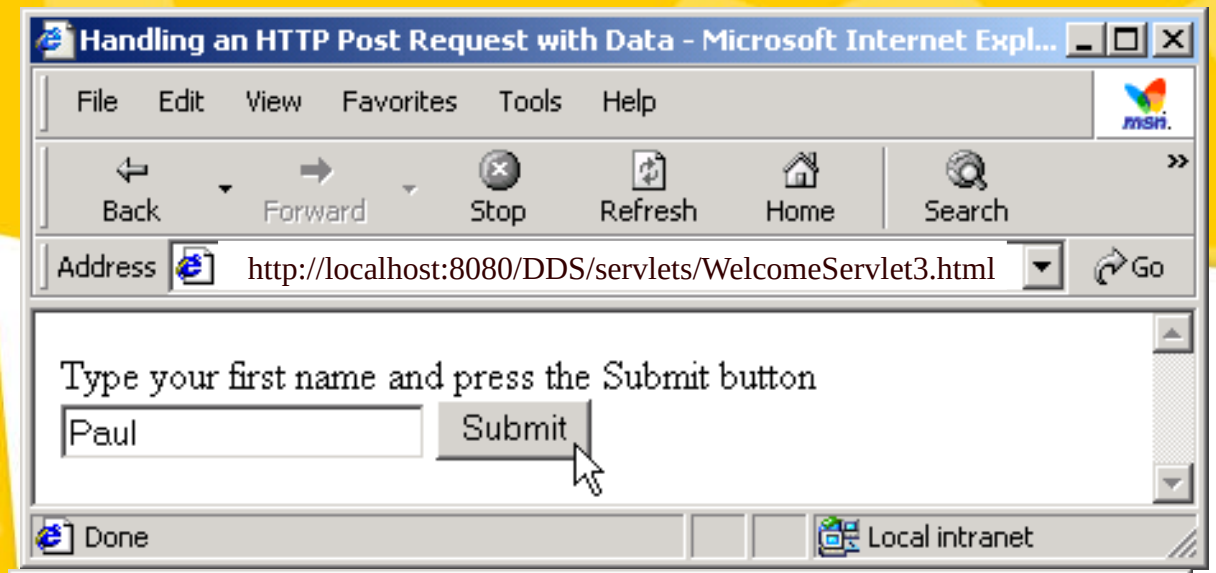
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- WelcomeServlet3.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9    <title>Handling an HTTP Post Request with Data</title>
10 </head>
11
12 <body>
13   <form action = "/DirectoryDiSaluto/welcome3" method = "post">
14
15     <p><label>
16       Type your first name and press the Submit button
17       <br /><input type = "text" name = "firstname" />
18       <input type = "submit" value = "Submit" />
19     </label></p>
20
21   </form>
22 </body>
23 </html>
```

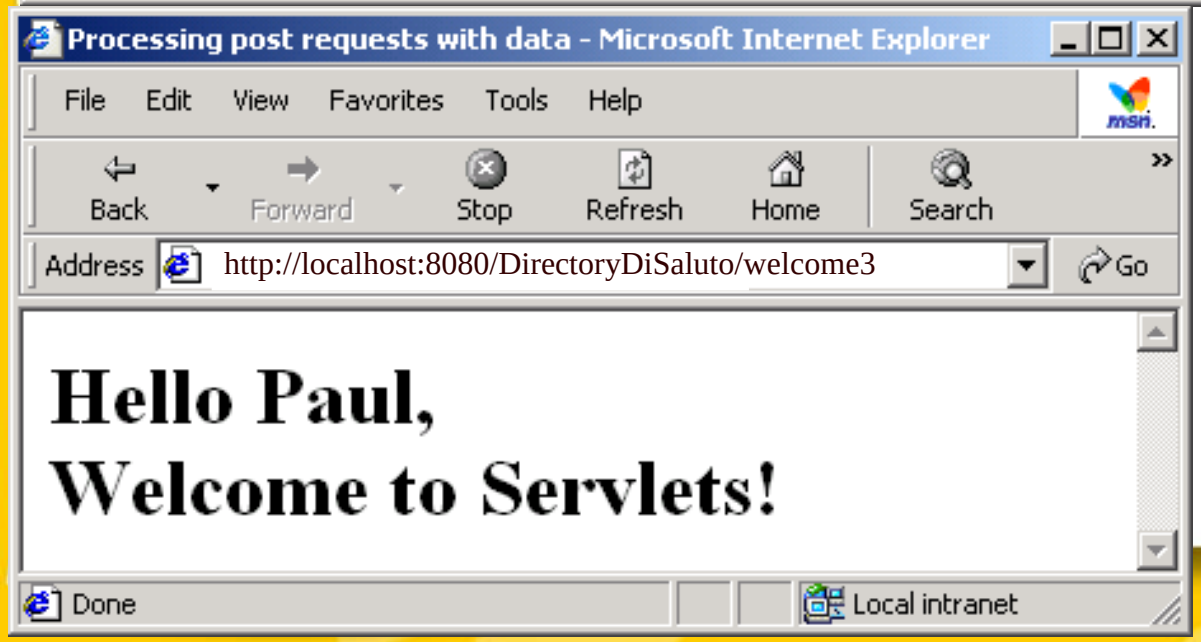
Viene fornito un **form** in cui l'utente può scrivere il proprio nome nell'elemento di input **firstname** di tipo **text**. Quando viene cliccato il bottone **Submit** viene invocata **WelcomeServlet3**.

```
2  // Processing post requests containing data.
3
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class WelcomeServlet3 extends HttpServlet {
10
11    // process "post" request from client
12    protected void doPost( HttpServletRequest request,
13      HttpServletResponse response )
14        throws ServletException, IOException
15    {
16      String firstName = request.getParameter( "firstname" );
17
18      response.setContentType( "text/html" );
19      PrintWriter out = response.getWriter();
20
21      // send XHTML page to client
22
23      // start XHTML document
24      out.println( "<?xml version = \"1.0\"?>" );
25
26      out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
27        "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
28        "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
29
30      out.println(
31        "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
32
```

Define a **doPost** method to respond to post requests.

```
33      // head section of document
34      out.println( "<head>" );
35      out.println(
36        "<title>Processing post requests with data</title>" );
37      out.println( "</head>" );
38
39      // body section of document
40      out.println( "<body>" );
41      out.println( "<h1>Hello " + firstName + ",<br />" );
42      out.println( "Welcome to Servlets!</h1>" );
43      out.println( "</body>" );
44
45      // end XHTML document
46      out.println( "</html>" );
47      out.close();  // close stream to complete the page
48   }
49 }
```


Handling an HTTP Post Request with Data - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DDS/servlets/WelcomeServlet3.html
Go
Type your first name and press the Submit button
Paul
Submit
Done
Local intranet


Processing post requests with data - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://localhost:8080/DirectoryDiSaluto/welcome3
Go
Hello Paul,
Welcome to Servlets!
Done
Local intranet

Modifiche al file web.xml

Descriptor element	Value
servlet *element*	
servlet-name	**welcome3**
description	**Handling HTTP post requests with data.**
servlet-class	**WelcomeServlet3**
servlet-mapping *element*	
servlet-name	**welcome3**
url-pattern	**/welcome3**

Non dimenticate di ricaricare l'applicazione dopo avere ultimato e salvato le modifiche al file web.xml