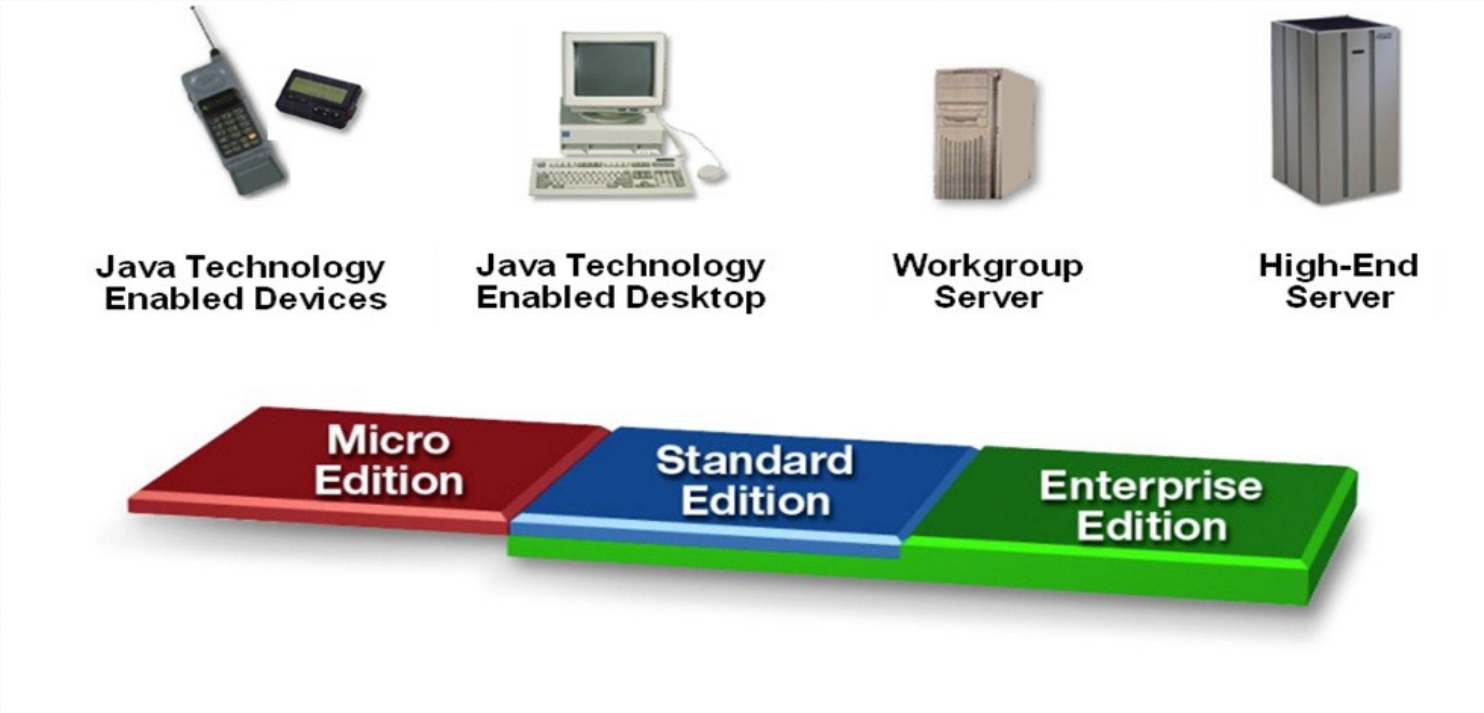
Java J2EE
JSP e Servlets

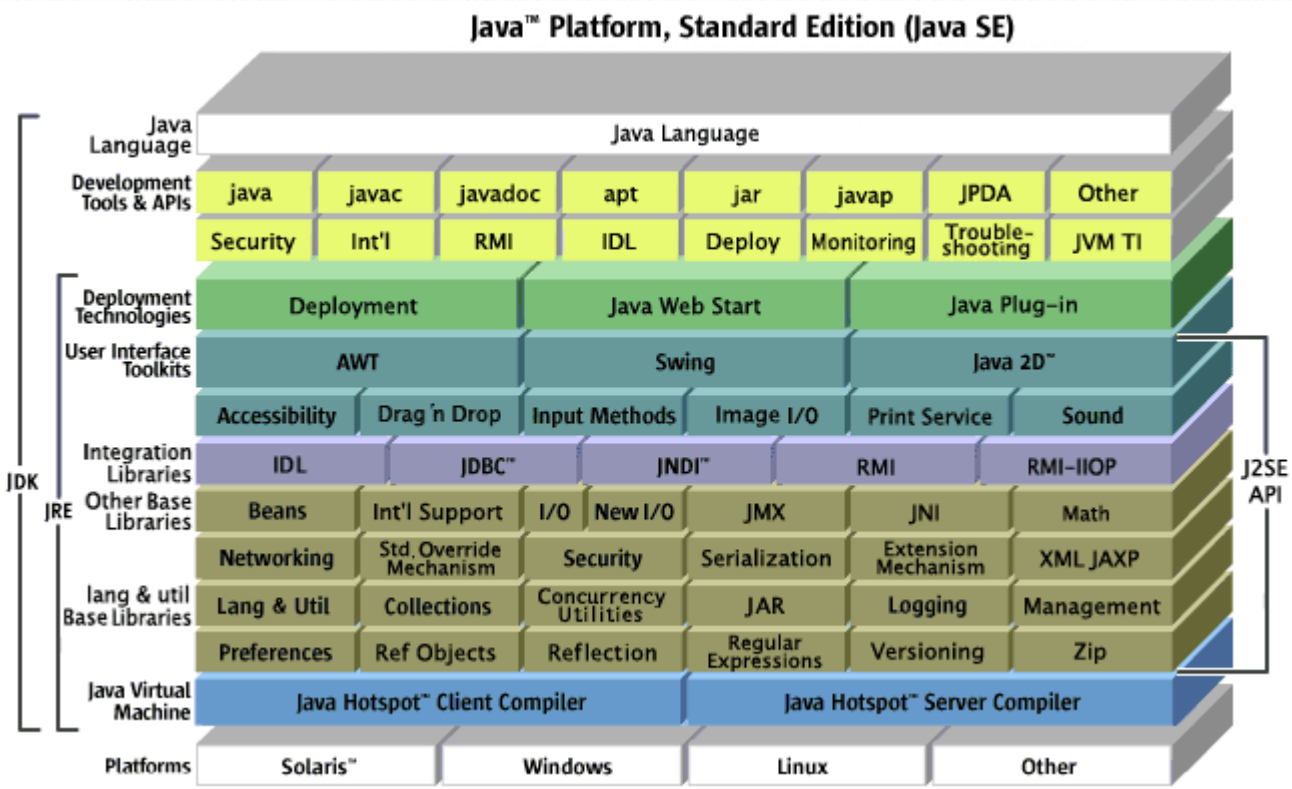
Relatore
Dott.ssa Lucrezia Macchia
Ph.D Student in Computer Science

Dipartimento DI INFORMATICA

Piattaforme JAVA



Ambienti di sviluppo: JDK



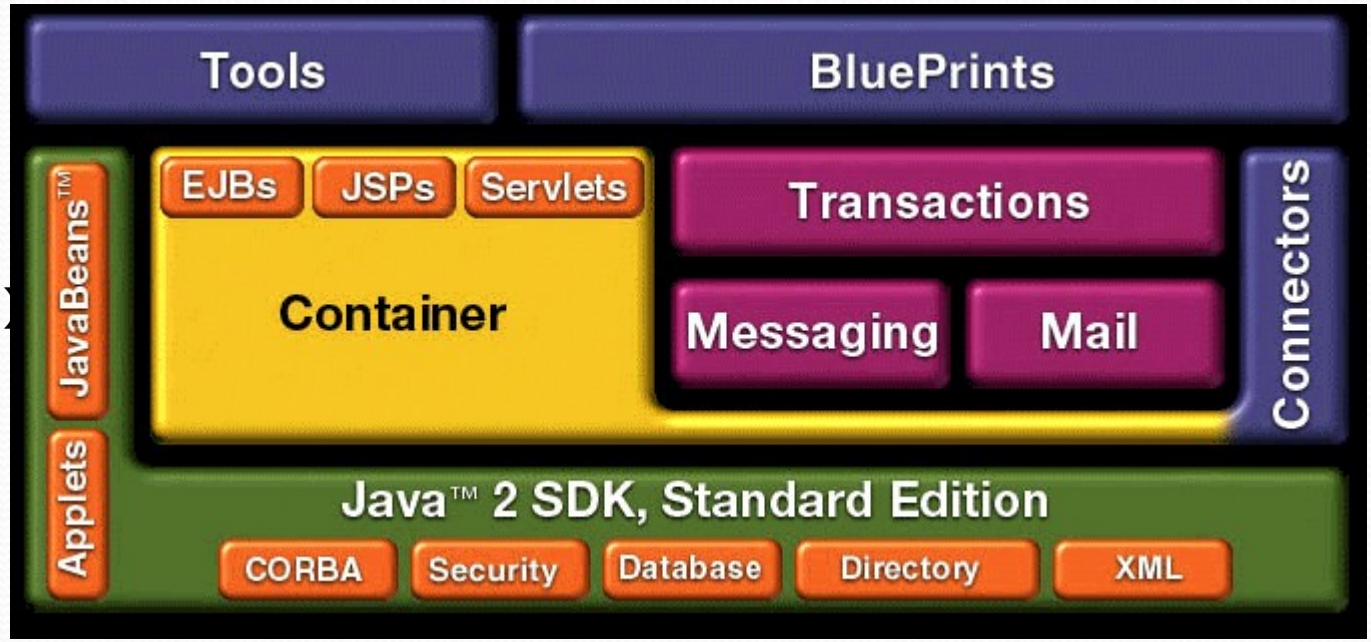
L' edizione SDK si distingue in:
– Java 2 Standard Edition J2SE
– Java 2 Entreprise Edition J2EE
– Java 2 Micro Edition J2ME
La prima (J2SE) rappresenta la piattaforma Java standard e include tutti gli applicativi e le librerie per lo sviluppo di applicazioni desktop, applet e client-server.

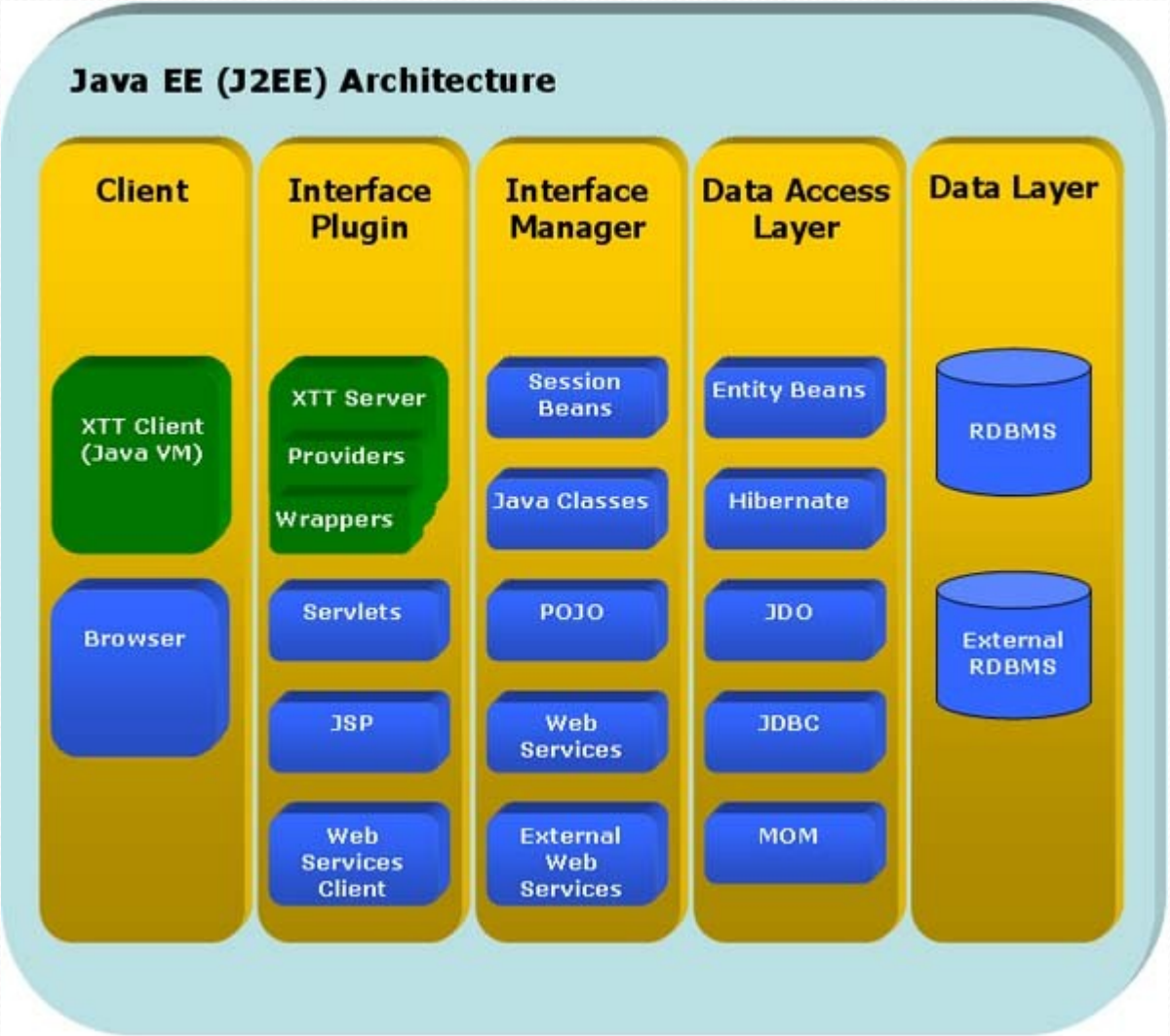
JAVA EE è un ambiente incentrato su Java indipendente dalla piattaforma per sviluppare, creare e distribuire le applicazioni aziendali online basate sul Web. Java EE include molti componenti di Java Standard Edition (Java SE).

La J2EE estende la J2SE includendo API progettate per lo sviluppo di applicazioni lato server che facciano uso ,tra le altre, delle seguenti tecnologie:

- **Enterprise JavaBeans (EJB),**
- **Servlets,**
- **JSP**

(Java Server Pages)




Java EE (J2EE) Architecture
Client
Interface Plugin
Interface Manager
Data Access Layer
Data Layer
XTT Client (Java VM)
Browser
XTT Server
Providers
Wrappers
Servlets
JSP
Web Services Client
Session Beans
Java Classes
POJO
Web Services
External Web Services
Entity Beans
Hibernate
JDO
JDBC
MOM
RDBMS
External RDBMS

Principali tecnologie:

JSP: JavaServer Pages è una tecnologia J2EE per costruire applicazioni web a contenuto dinamico, come HTML, DHTML, XHTML ed XML

SERVLET: è un componente web scritto in Java, gestito da un "container", che produce contenuto dinamico.

EJB: le specifiche EJB intendono fornire una metodologia standard per implementare la logica di funzionamento delle applicazioni di tipo *enterprise*

System Requirements:

IDE J2EE :

esempio: Eclipse Java EE

SDK:

Java Platform, Enterprise Edition 6 SDK

Servlet Engine :

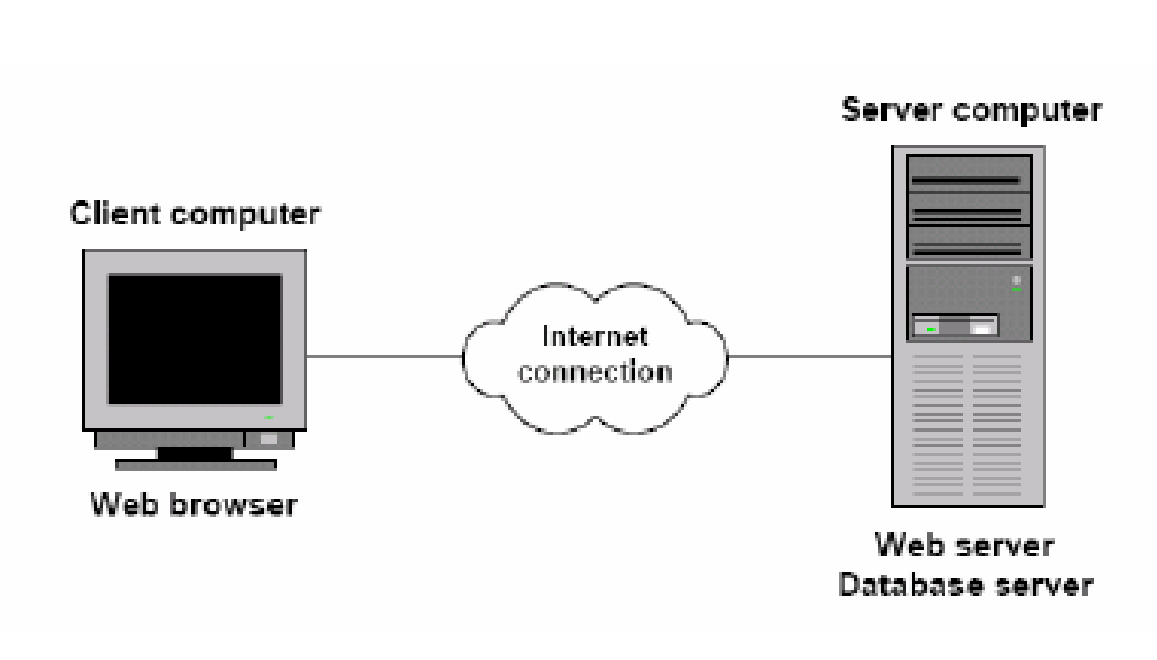
esempio: Tomcat

Il contesto: Web

- **Un'applicazione Web è un'applicazione client/server a cui è possibile accedere mediante un browser web.**

- **L'utente da un comando al browser. Ad esempio: indicando l'indirizzo della pagina web che vogliamo visualizzare;**

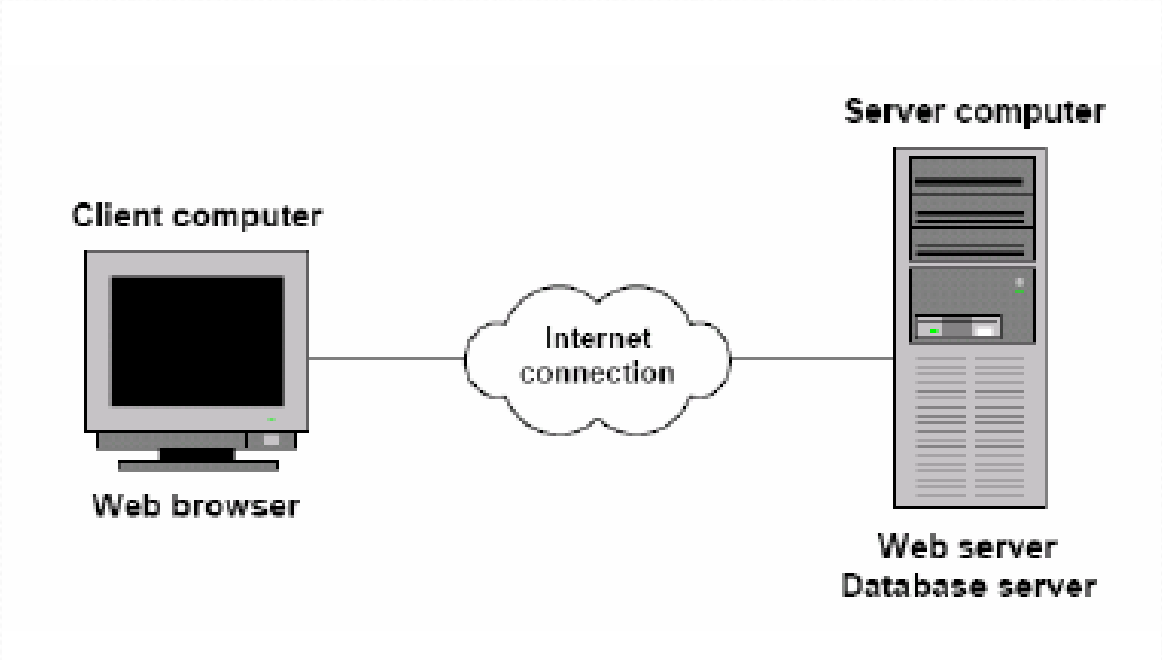
- **Il browser si collega al server corrispondente all'indirizzo inserito;**



Il contesto: Web

- **Il server elabora la pagina richiesta e restituisce al browser i dati che compongono la pagina;**
- **Il browser riceve i dati dal server e li mostra all'utente.**

- **L'applicazione web è caratterizzata da un insieme di pagine generate in risposta alle richieste degli utenti**



- **Esistono principalmente due strade attraverso le quali possiamo pubblicare contenuti destinati al Web:**

1. **Creare una per una le singole pagine che andranno a comporre il sito internet utilizzando del semplice codice HTML.**
2. **Utilizzare una tecnologia *Server Side* che permetta di produrre dinamicamente codice HTML attraverso il quale rendere disponibili i contenuti.**

- **Nel primo caso si parla di "pagine web statiche"**
- **Nel secondo caso parliamo invece di "pagine web dinamiche", o meglio di pagine web "generate dinamicamente".**

- **Un sito composto da pagine "statiche" non è certo comodo da gestire,**
  - **bisogna impaginare e formattare "a mano" i nuovi contenuti e così anche tutto ciò che gravita attorno ad essi: layout, menu, eventuali mappe..**
- **Utilizzando il semplice HTML sarà poi arduo mettere a disposizione degli utenti servizi addizionali come forum di discussione, newsletter, guestbook...**
- **L'utilizzo di tecnologie *Server Side* per la produzione "dinamica di contenuti" come PHP, ASP e JSP, ci consente di costruire**
  - **siti professionali semplici da gestire, compatibili con i più diffusi browser esistenti e soprattutto capaci di "interagire" con gli utenti soddisfacendo le loro richieste sulla base del meccanismo "input/output", "domanda/risposta".**

Le pagine Web concepite secondo tecnologie *Server Side* funzionano in linea generale sempre nello stesso modo:

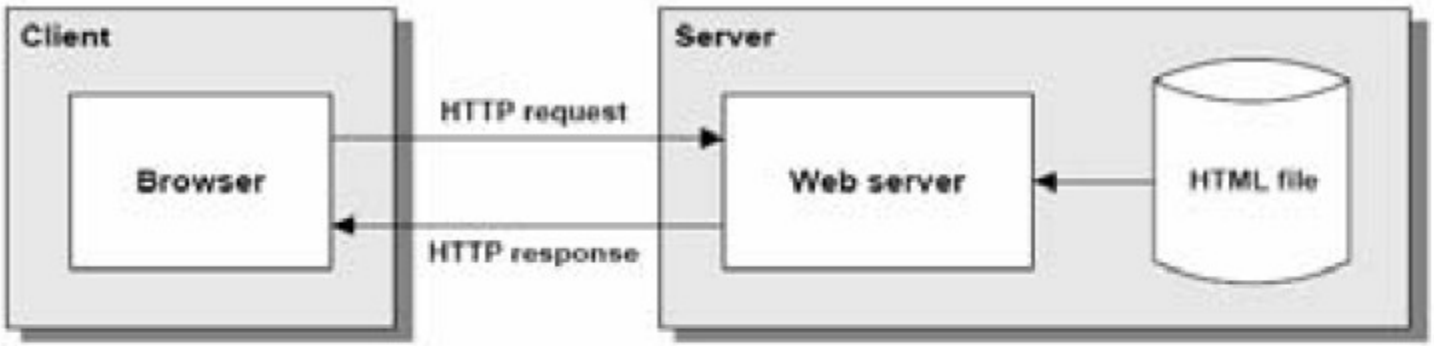
1)L'utente, attraverso il suo *browser* invia una richiesta

2)La richiesta viene raccolta e interpretata da un Web server che ricerca la risorsa desiderata all'interno del server in cui è installato.

3)Trovata la risorsa richiesta, il Web Server la trasmette al *browser* del *client* sotto forma di codice HTML.

4)La tecnologia per le JSP consente la creazione di "pagine dinamiche" capaci di interloquire da una parte con l'utente e dall'altra con un Web Server, o per meglio dire con Servlet Engine JSP come per esempio Tomcat.

Pagine statiche

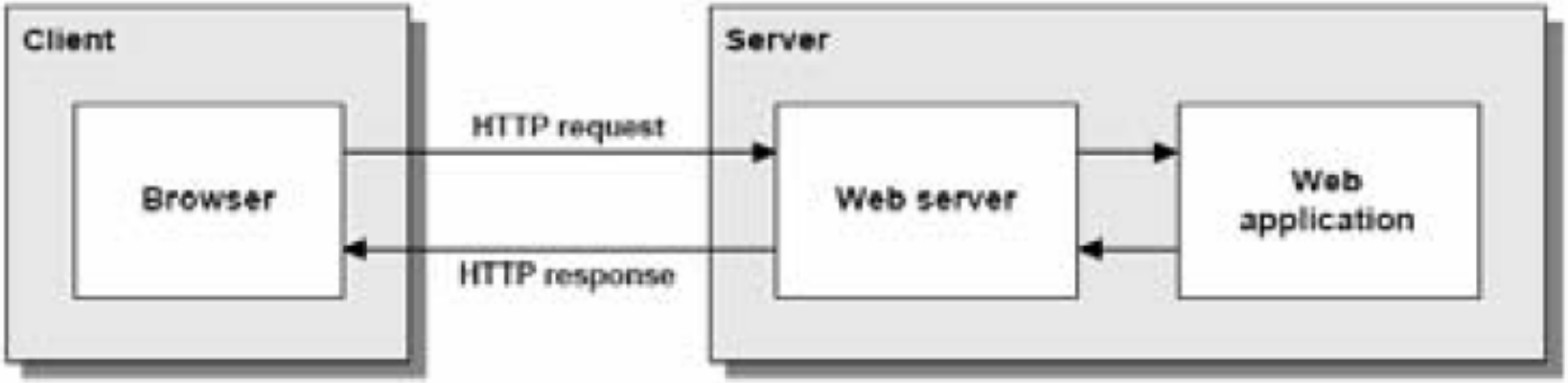


- Le pagine web statiche hanno nomi con estensione .htm o .html.

Se si richiedono pagine web statiche la HTTP request include il nome del file html richiesto

- Il web server risponde con una HTTP response che include il file HTML

Pagine dinamiche

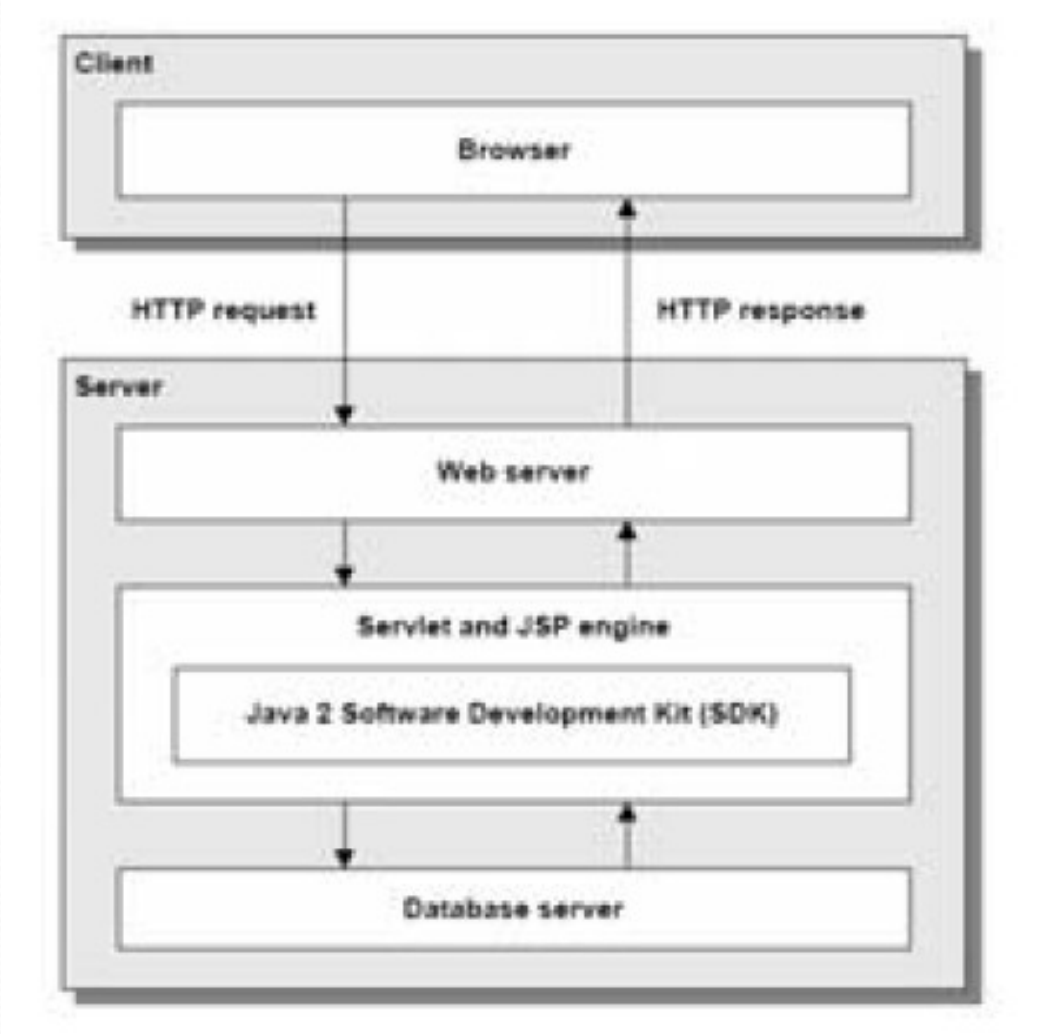


- Una pagina *web dinamica è* un file HTML generato da una web application. Spesso queste pagine cambiano in base a dei parametri che vengono inviati dal browser alla web application.

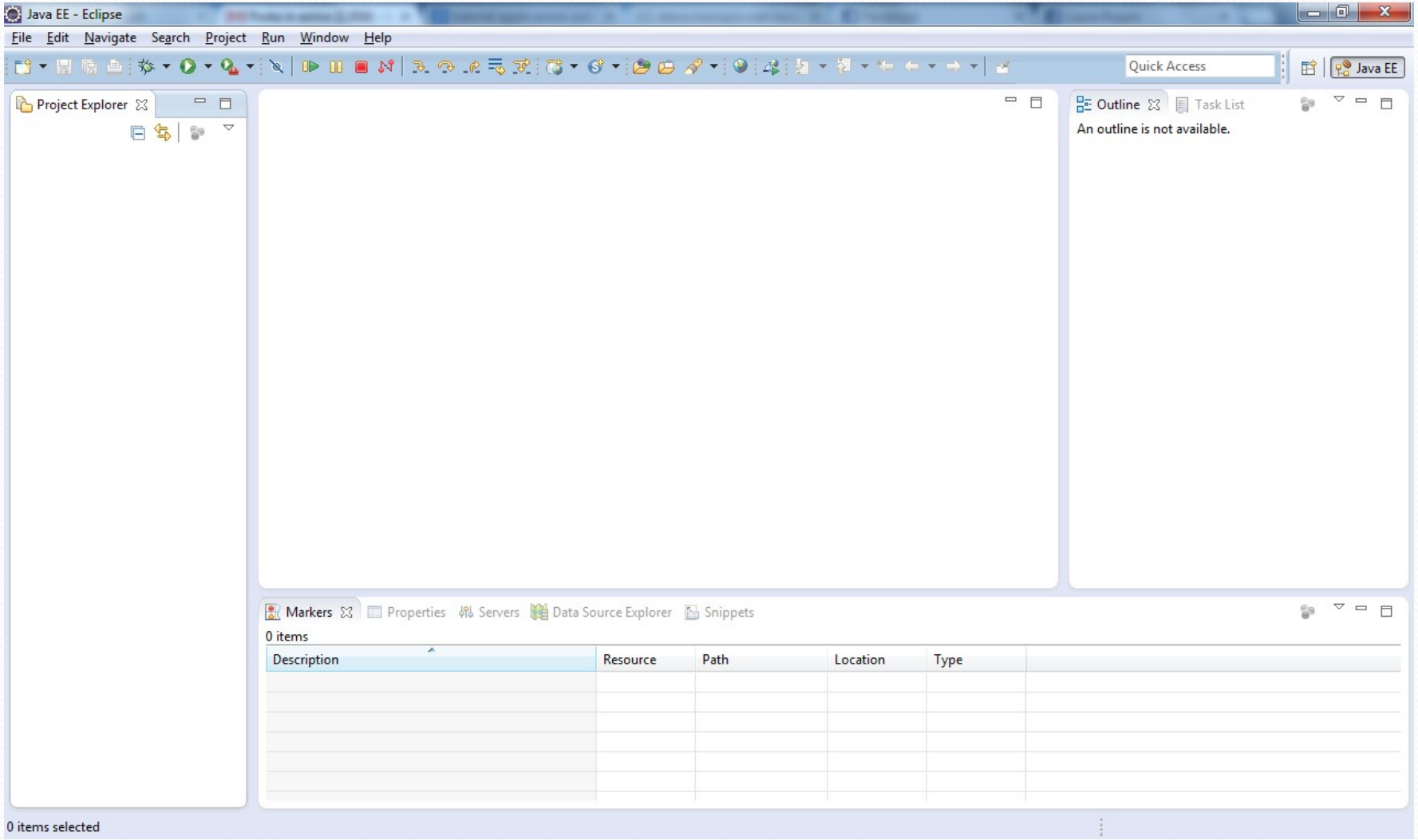
- Quando un web server riceve la richiesta di una pagina web dinamica, il server inoltra la richiesta alla web application che genera una risposta sotto forma di file HTML.

Il web server invia al browser il documento HTML generato in una HTTP response. Il browser ignora se il file HTML è stato generato dinamicamente o meno.

Riassumendo: Java Web Application



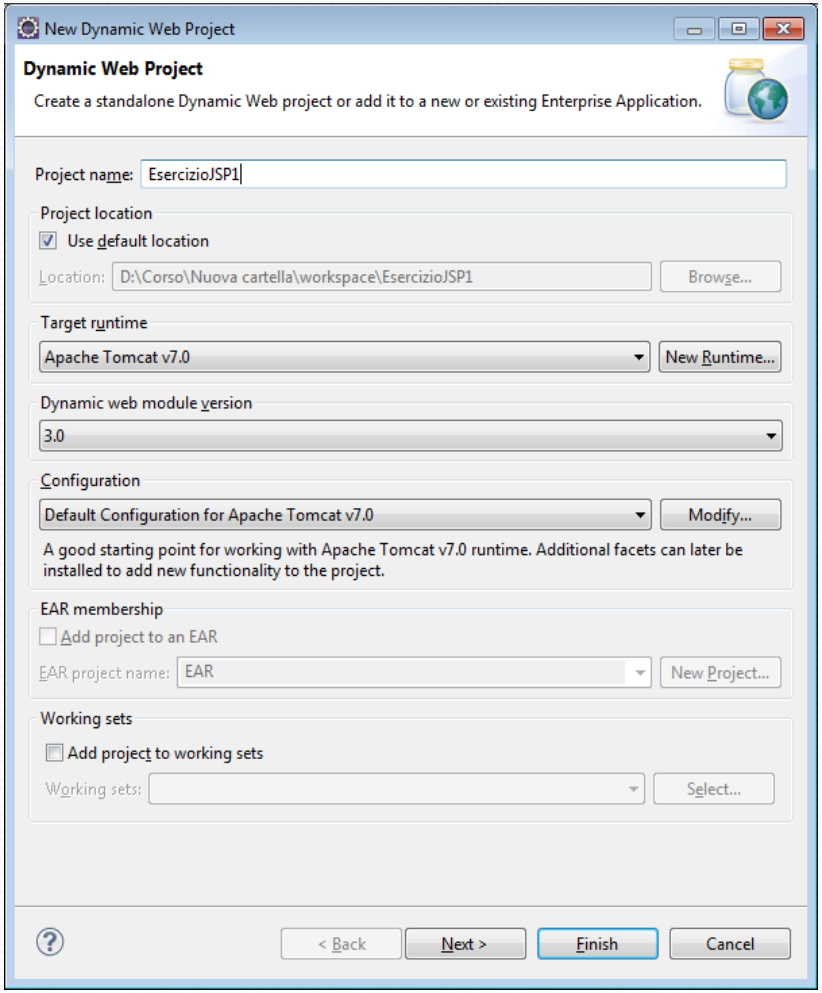
Esempio: IDE e prima pagina jsp

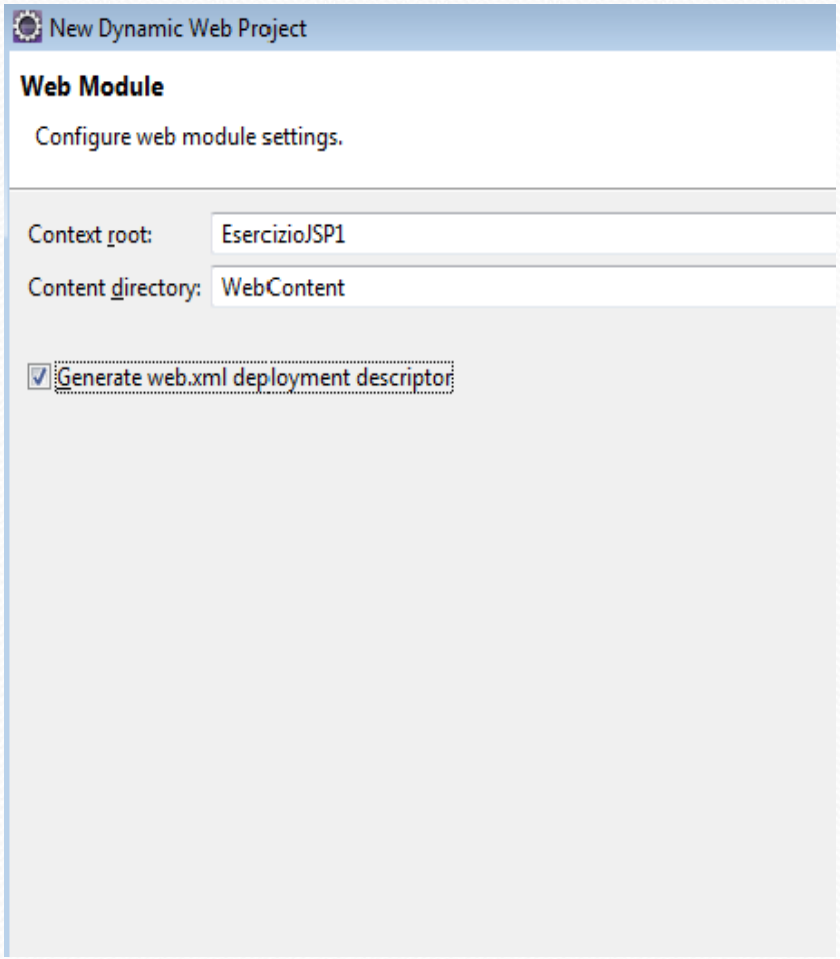


Creazione Nuovo Progetto:

Selezioniamo la voce di menu File>New

Selezioniamo la voce “Dynamic Web Project” e clicchiamo sul pulsante “Next”



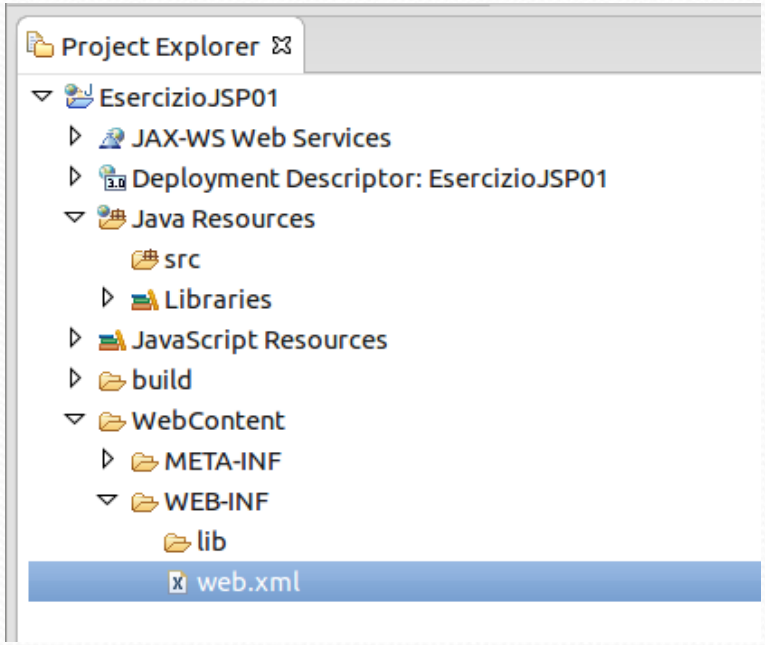


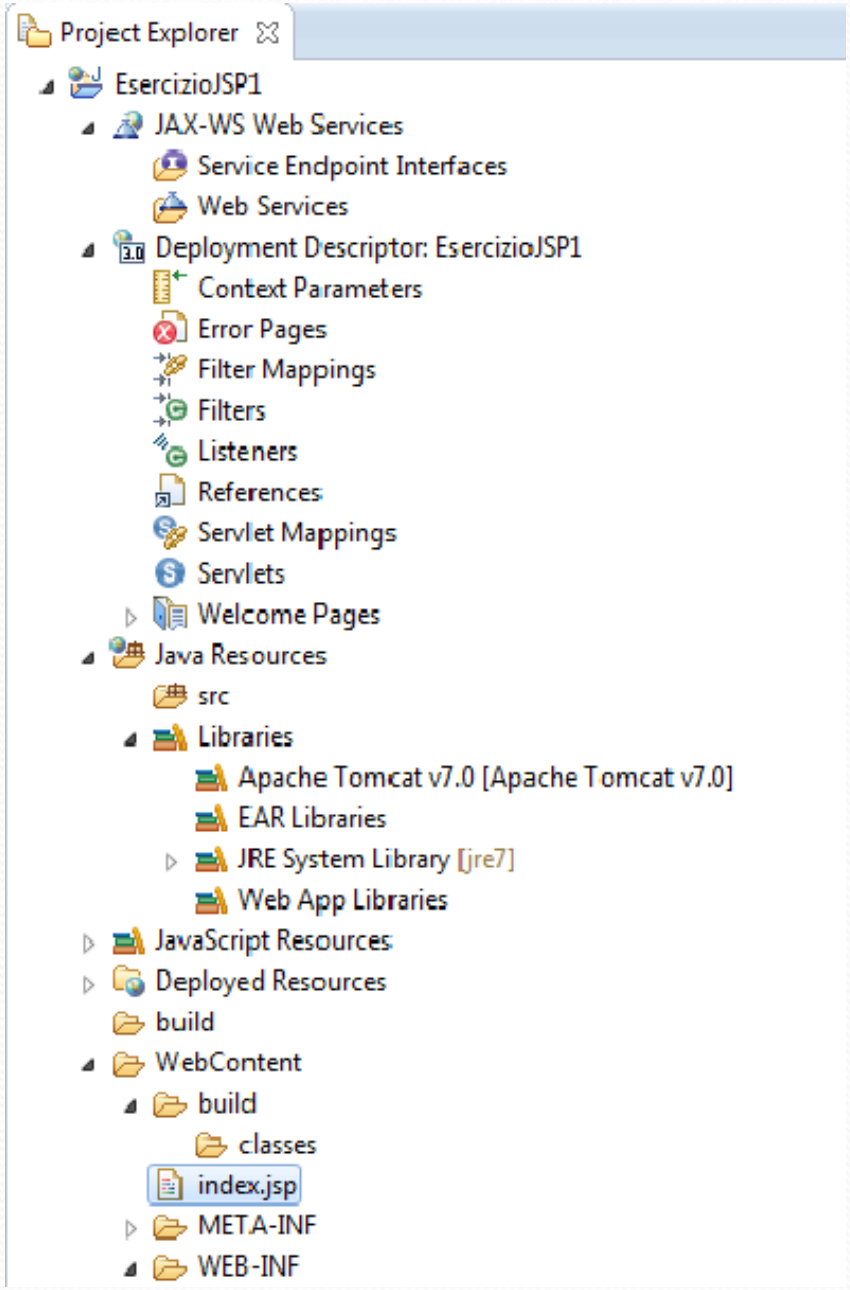
Esempio 1

Stampare a video la scritta "Prova"

Effettuare il deploy del progetto e fare l'upload su Tomcat

Appena avviata l'applicazione web deve aprire la pagina index.jsp


Project Explorer
EsercizioJSP01
JAX-WS Web Services
Deployment Descriptor: EsercizioJSP01
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
web.xml


Project Explorer
EsercizioJSP1
JAX-WS Web Services
Service Endpoint Interfaces
Web Services
Deployment Descriptor: EsercizioJSP1
Context Parameters
Error Pages
Filter Mappings
Filters
Listeners
References
Servlet Mappings
Servlets
Welcome Pages
Java Resources
src
Libraries
Apache Tomcat v7.0 [Apache Tomcat v7.0]
EAR Libraries
JRE System Library [jre7]
Web App Libraries
JavaScript Resources
Deployed Resources
build
WebContent
build
classes
index.jsp
META-INF
WEB-INF

*web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="htt
  <display-name>Esercizio1</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Pagina JSP</title>
</head >
<body>
<% out.println("<h1>Prova</h1>"); %>
</body>
</html>
```

- **Il codice Java della pagina "index" è stato isolato tra i delimitatori "<%" e "%>"(chi ha pratica con ASP li avrà sicuramente utilizzati tante volte) in modo da poter essere riconosciuto come tale.**
- **L'istruzione out.println ha il compito di stampare a video il parametro che le viene passato**

JSP:Definizione

JavaServer Pages è una tecnologia J2EE per costruire applicazioni web a contenuto dinamico, come HTML, DHTML, XHTML ed XML.

JSP permette di creare pagine web a contenuto dinamico con semplicità e flessibilità

Che cos'è una JSP Page?

Una pagina JSP è un documento di testo che descrive come elaborare una richiesta (request) per generare una risposta (response).

- **“Ciao Mondo!” da stampare a video.**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loos
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Prova.jsp</title>
</head>
<body>
<% out.println("Ciao Mondo!"); %>
</body>
</html>
```

- **Come si può notare le sezioni racchiuse tra <% e %> sono quelle che contengono le istruzioni in linguaggio Java**

- ***Nota: i file per essere riconosciuti dal server come pagine JSP devono essere salvati con estensione .jsp.***

Gli elementi fondamentali che si presentano spesso all'interno dello scripting JSP:

- **Le Dichiarazioni.**
- **Le Scriptlet.**
- **Le Espressioni.**
- **Le Actions o Standard Actions**

- Le Dichiarazioni vengono espresse sintatticamente in questo modo:

```
<%! codice %>
```

attraverso questo semplice script:

```
<%! private int accessCount = 0; %>

<%= ++accessCount %>
```

Si crea un efficiente contatore di accessi per le pagine Web. Nella prima riga è stata espressa la dichiarazione e il nostro contatore partirà da zero, successivamente, nella seconda riga indichiamo un incremento unitario ("++") per ogni accesso rilevato.

```
<% java.util.Date data_di_oggi = new java.util.Date();
out.println( String.valueOf( data_di_oggi )); %>
```

- **Testando le poche righe di codice appena digitate, visualizzeremo in output la data odierna completa di giorno, mese, anno, ore, minuti e secondi.**

- **L'uso delle Espressioni da parte dello sviluppatore viene facilitato dall'esistenza di oggetti predefiniti che possono essere immediatamente utilizzati per produrre il comportamento per cui sono deputati. Ad esempio:**

```
<%= request.getRemoteHost() %>
```

sfrutta l'oggetto request per richiedere il nome di Host del server (se testiamo lo script su Tomcat in locale a meno di nostre modifiche otterremo il classico 127.0.0.1).

- Le **Actions** o **Standard Actions** sono tags specifici che agiscono sul comportamento dei file ".jsp" e sugli output destinati a soddisfare le richieste

  - <html> <head> <title>jsp:include</title> <b>Output<b>: <jsp:include page="index.jsp"/> </body> </html>

  - <html> <head> <title>jsp:forward</title> <b>Output<b>: <jsp:forward page=" index.jsp "/> </body> </html>

il primo "**includerà**" l'output previsto in "<index.jsp" all'interno di "jsp_prova.jsp"

- il secondo "**inoltra**" l'output di "jsp_prova.jsp" in "index.jsp".

- **Le Direttive non producono output ma mettono a disposizione definizioni, notifiche e informazioni valide e disponibili in tutto il documento ".jsp". La sintassi più utilizzata per le Direttive è:**

<%@ Codice %>

- **Esistono tre tipologie principali di Direttive utilizzabili:**
  - **Page: attraverso la quale vengono direttamente coinvolti i criteri di compilazione delle pagine.**
  - **Include: include stringhe di testo o ulteriore codice all'interno di un file.**
  - **Taglib: abbreviazione di *tag library*, attraverso di essa possiamo utilizzare tags personalizzati, indicare la libreria in cui sono definiti e associare loro un prefisso che li renda univoci.**

Direttiva Page

esprime una "definizione degli attributi"

```
<%@ page import="java.util.*" %>
<html> <head>
 <% Date data_di_oggi = new Date(); %>
 <%= data_di_oggi %>
</head> </html>
```

è usata per importare il *package* java.util, per introdurre ulteriori *packages* si intervallano con una virgola.

Direttiva Include

- esprime una "notifica" al Servlet Engine riguardo a un punto determinato del file ".jsp" in cui và incluso del codice esterno

- <html> <head>
- <b>Output<b>: <%@ include file="jsp_prova.jsp" %>
- </head> </html>

- Si potrebbe includere qualsiasi tipo di file

Taglib

"autorizza" l'uso di costrutti sintattici definiti dallo sviluppatore

```
<%@ taglib uri="tag library" prefix="prefisso per i tag" %>
```

L'uri indica l'archivio della *tag library* scelto dallo sviluppatore, mentre prefix indica il prefisso utilizzato per il richiamo dei tags.

Un esempio dell'uso di Taglib potrebbe essere questo:

```
<%@ taglib uri="http://www.sito.it/tags" prefix="tgs" %>
```

```
<tgs:Tag attribute="value"> STRINGA </tgs:Tag>
```

Oggetti Impliciti

- **Nella creazione di pagine ".jsp" possiamo avvalerci di differenti oggetti impliciti:**

- **request:**

  istanza della classe HttpServletRequest associata alla richiesta di risorse generata in input dall'utente;

  grazie ad essa abbiamo la possibilità di rilevare parametri fondamentali (getParameter) come: metodi utilizzati per la richiesta (Post, Get..), intestazioni (headers..)

- **response:**

  istanza della classe HttpServletResponse associata alla risposta generata in output dal Servlet Engine;

  attraverso questa variabile possiamo notificare errori, spedire intestazioni, settare cookies, etc...

- **application**:

  il ServletContext generato sulla base dell'azione operata sui parametri passati a getServletConfig().getContext(); permette di accedere al contesto in cui viene eseguita la Web application, sarà quindi possibile ottenere informazioni sull'Engine JSP e reperire parametri inizializzati.

- **config**:

  la ServletConfig riferita alla pagina;

- **pageContext**:

  istanza della classe PageContext

  l'oggetto pageContext è in grado di rappresentare ogni elemento riconducibile al contesto in cui viene eseguita la pagina.

- **out:**

  il cosiddetto PrintWriter, a cui spetta la funzione fondamentale di inviare l'ouput al *client* da cui è stata spedita la richiesta attraverso il protocollo di comunicazione.

- **session:**

  HttpSession associato alla richiesta; ci consente d'identificare l'utente rendendo la sua sessione valida per tutto l'ambito di un'applicazione

- **Per evitare di riempire le pagine Jsp di centinaia di righe di codice e renderle quindi difficilmente manipolabili, c'è un modo per spostare totalmente sul server la parte di calcolo.**
- **Il concetto è il seguente:**
  - **quando il browser richiede di visualizzare una pagina Jsp**
  - **il server, individuato il file richiesto, interpreta il codice Java contenuto nella pagina ed invia al browser solamente la pagina HTML con il risultato.**

E' preferibile spostare il codice in una classe Java e farla semplicemente richiamare dalla pagina Jsp. Cosicché il controllo verrebbe comunque fatto dal server, ma la progettazione risulta più semplice e scalabile.

JSP
Benefici offerti

- "Write once run anywhere"
- JSP supporta e incoraggia l'uso di componenti riutilizzabili e cross platform come

  JavaBeans,

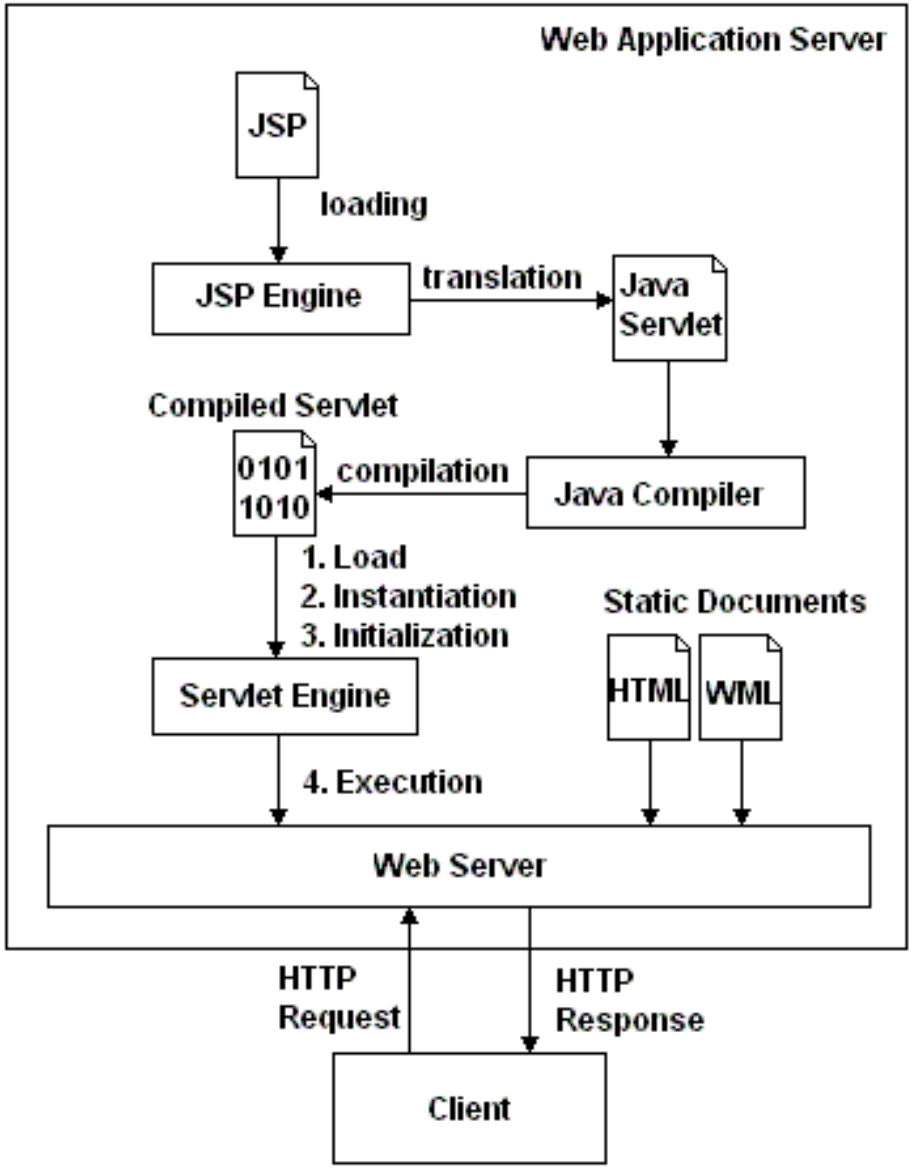
  tag libraries,

  EJB.
- JSP è parte integrante di J2EE ed è uno standard ampiamente supportato da tutti gli application server J2EE.
- E' supportato dai tool di design web

JSP
Benefici offerti

- **Favorisce la “separazione dei ruoli”:**
  - **gli sviluppatori scrivono i componenti che interagiscono col server,**
  - **i web authors uniscono dati statici e dinamici per realizzare l’applicazione web.**
- **Incoraggia la separazione tra contenuti statici e contenuti dinamici**

JSP: funzionamento



JSP Scripting Elements

Gli Scripting Elements sono usati per includere codice (anche non Java) nelle JSP.

Declarations:

```
<%! int i = 4 %>
```

Sono usate per dichiarare variabili e metodi in linguaggio di scripting di una pagina.

Scriptlets:

```
<% for (int i=0; i10; i++) {
          out.println("The counter is:" + i);
   } %>
```

Sono utilizzate per contenere qualsiasi frammento di codice che è valido per il linguaggio di scripting utilizzato

Expressions:

```
<%= myBean.getNumber() %>
```

Sono usate per inserire il valore di un'espressione in linguaggio di script, convertito in stringa, nel flusso di dati restituito al client

JSP Standard Actions

JSP Standard Actions sono tag XML specifici che influenzano l'esecuzione della pagina e l'output spedito al client.

Esempi:

```
<jsp:useBean id="myBean" scope="application" class="mypackage.MyBean" />

<jsp:setProperty name="myBean" property="age" />

<jsp:include page="filename" />

<jsp:forward page="url" />

<jsp:param name="paramName" value="paramValue" />
```

- `<jsp:useBean id="myBean" scope="application" class="mypackage.MyBean" />`

***<jsp:useBean>*: associa un'istanza di un JavaBean a un ambito e ad un ID tramite una variabile di script dichiarata con lo stesso nome; se non trova un'istanza "in vita" cerca di crearla. Gli attributi più importanti legati a questa azione sono:**

- ***id* : rappresenta l'identità dell'istanza dell'oggetto all'interno dell'ambito specificato;**
- ***scope* : rappresenta l'ambito dell'oggetto. Possibili scope: page, request, session, application;**
- ***class* : rappresenta il nome di classe che definisce l'implementazione dell'oggetto.**

```
<jsp:setProperty name="myBean" property="age" />
```

- **questa azione principale derivano le seguenti sottoazioni:**
  - ***<jsp:setProperty>*: imposta il valore di una proprietà di un bean;**
  - ***<jsp:getProperty>*: prende il valore della proprietà del bean referenziato, lo converte in un java.lang.String e lo inserisce nell'oggetto out implicito;**

- **Un JavaBeans è un componente Java al 100% che opera all'interno di una qualsiasi macchina virtuale; è una classe Java che implementa l'interfaccia java.io.Serializable e utilizza metodi pubblici get/set per esporre le proprietà.**

- `<jsp:include page="filename" />`
- ***<jsp:include>*: fornisce un modo per includere risorse aggiuntive, di tipo statico o dinamico, nella Pagina JSP corrente. Gli attributi più importanti legati a questa azione sono:**
  - ***page* : rappresenta la URL relativa della risorsa da includere;**
  - ***flush* : questo attributo contiene un valore boolean che indica se il buffer debba o meno essere svuotato.**

- `<jsp:forward page="url" />`

- ***<jsp:forward>*: consente al motore Jsp l'inoltro, all'atto dell'esecuzione, della risorsa corrente a una risorsa statica, a un Servlet o ad una Pagina JSP. L'unico attributo possibile per questa azione è:**
  - ***page* : rappresenta la URL relativa all'oggetto da inoltrare.**

- `<jsp:param name="paramName" value="paramValue" />`
- ***<jsp:param>*: viene impiegata per fornire informazioni a coppie di tag/valore, includendoli come sottoattributi delle azioni *<jsp:forward>*, *<jsp:useBean>*, *<jsp:include>* all'atto dell'esecuzione, della risorsa corrente a una risorsa statica, a un Servlet o ad una Pagina JSP. L'unico attributo è:**
  - ***name* : rappresenta il nome del parametro referenziato;**
  - ***value* : rappresenta il valore del parametro referenziato.**

Esempio 2

Creare una pagina web che permetta di inserire una categoria e un bottone che una volta cliccato reindirizzi il nome alla pagina jsp (news.jsp) che si occuperà di scrivere

Sezione XXX

Dove XXX rappresenta la categoria presa in in input

Le categorie possono esse: finanza, politica, sport

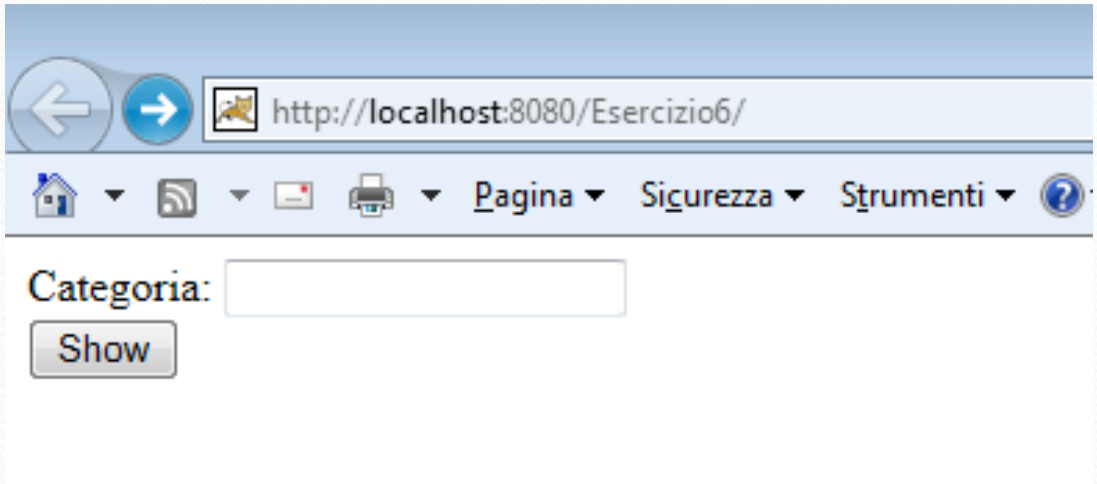
news.jsp | index.html

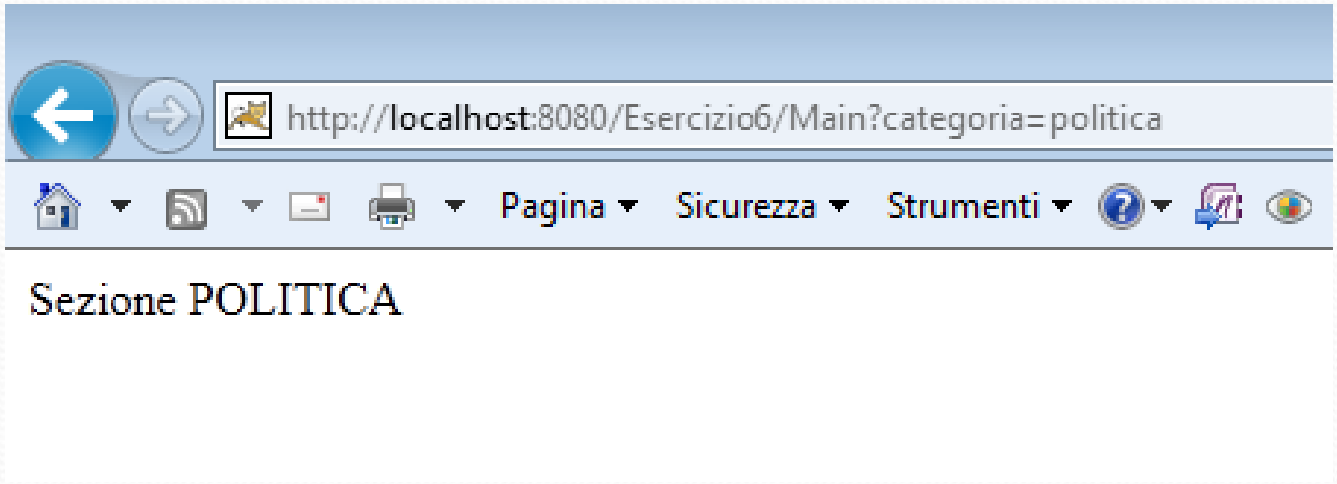
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loo
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>

  <body>
    <div style="text-align:center">
      <form name="Login" action="news.jsp">
        Categoria: <input name="categoria" type="text" /><br />
                <input type="submit" value="Show" />
      </form>
    </div>
  </body>

</html>
```

- EserciziojM2
  - JAX-WS Web Services
    - Loading...
  - Deployment Descriptor: EserciziojM2
  - Java Resources
    - src
    - Libraries
  - JavaScript Resources
  - Deployed Resources
  - build
  - WebContent
    - index.html
    - META-INF
    - news.jsp
    - WEB-INF
      - lib
      - web.xml

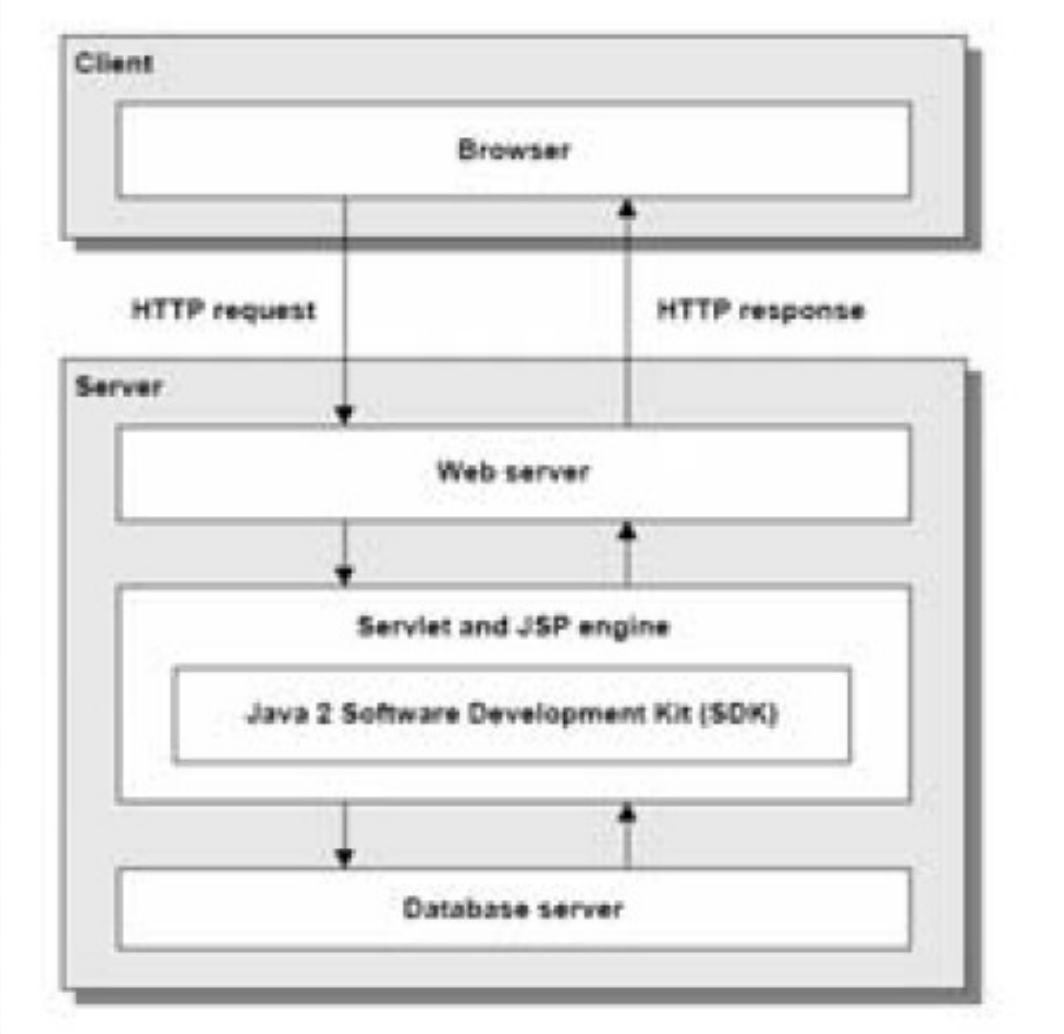
news.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htr
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<%
  if (request.getParameter("categoria").equals("politica")) {
%>
    <table style="width:200px">
      <tr><td style="text-align:center">Prima Sezione   POLITICA</td></tr>
    </table>
<%
  }
  else if (request.getParameter("categoria").equals("finanza")) {
%>
  <table style="width:200px">
      <tr><td style="text-align:center">Seconda Sezione   FINANZA</td></tr>
    </table>
<%
  }
  else if (request.getParameter("categoria").equals("sport")) {
%>
    <table style="width:200px">
      <tr><td style="text-align:center">Terza Sezione   SPORT</td></tr>
    </table>
<%
  }
%>
  </body>
</html>
```


http://localhost:8080/Esercizio6/
Pagina
Sicurezza
Strumenti
Categoria:
Show


http://localhost:8080/Esercizio6/Main?categoria=politica
Pagina
Sicurezza
Strumenti
Sezione POLITICA

Riassumendo: Java Web Application



Cos’è una Servlet?

con Servlet viene denominata un'applicazione, scritta in linguaggio Java, che è in esecuzione nella Java Virtual Machine.

È costantemente in attesa di soddisfare le chiamate che vengono raccolte dal Web server e passate al Servlet Engine JSP.

Più semplicemente:

Una Servlet è un componente web scritto in Java, gestito da un “container”, che produce contenuto dinamico.

Il compito delle Servlet, è quello di eseguire determinate operazioni con la finalità di produrre dinamicamente, in output, codice HTML da inviare come risposta al *browser* del *client* che ha effettuato la richiesta di una qualsiasi risorsa presente nel server

Nel caso specifico, al momento dell'invocazione di una pagina ".jsp" si succederanno le fasi seguenti:

- **Chiamata di una pagina ".jsp" tramite *browser*.**
- **Raccolta della chiamata da parte del Web server.**
- **Il Web server "vede" la pagina ".jsp", la riconosce e la consegna nelle mani dell'Engine JSP, ad esempio Tomcat.**
- **La pagina ".jsp" viene processata una prima volta.**
- **Il file ".jsp" viene tradotto in una classe Java e compilato in una Servlet(un'applicazione residente sul server).**

- **Vengono chiamati i metodi init() e service() in modo che la pagina ".jsp" non debba subire nuovamente lo stesso processo di riconoscimento, traduzione e compilazione ad ogni nuova chiamata.**

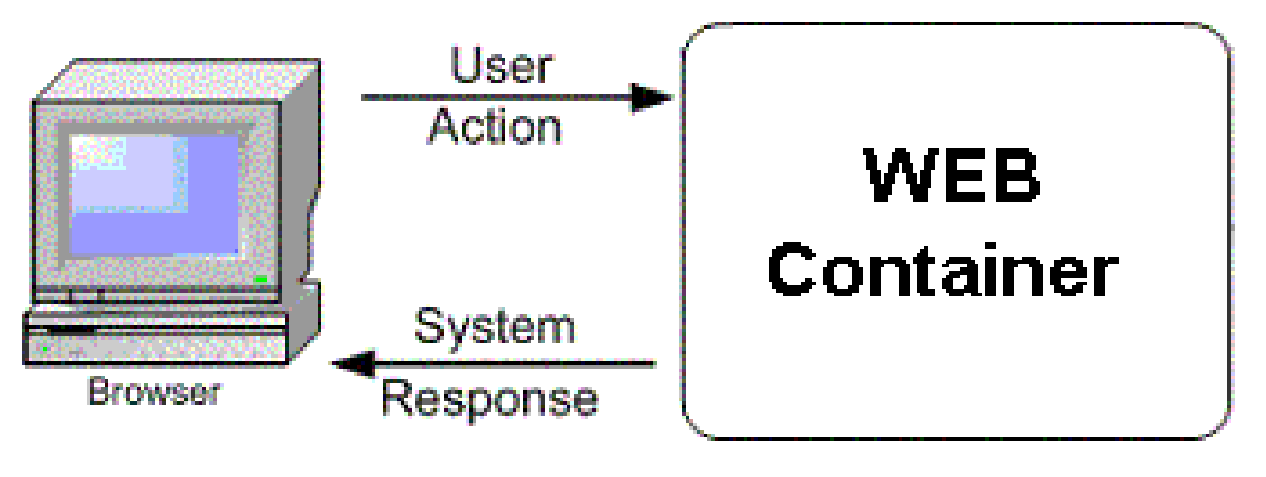
- **La richiesta effettuata attraverso la chiamata viene soddisfatta sotto forma di un output costituito da una pagina recante codice HTML prodotto dinamicamente.**

- **La Servlet ottenuta dalla traduzione e compilazione del codice per le JSP si avvale del metodo init(), quest'ultimo viene chiamato un prima volta quando la Servlet viene caricata, da quel momento essa sarà in esecuzione sulla JVM (Java Virtual Machine) sempre pronta a rispondere ad ogni successiva chiamata.**

Cos’è una Servlet?

Una Servlet interagisce con un web Client attraverso il paradigma di comunicazione

request/response



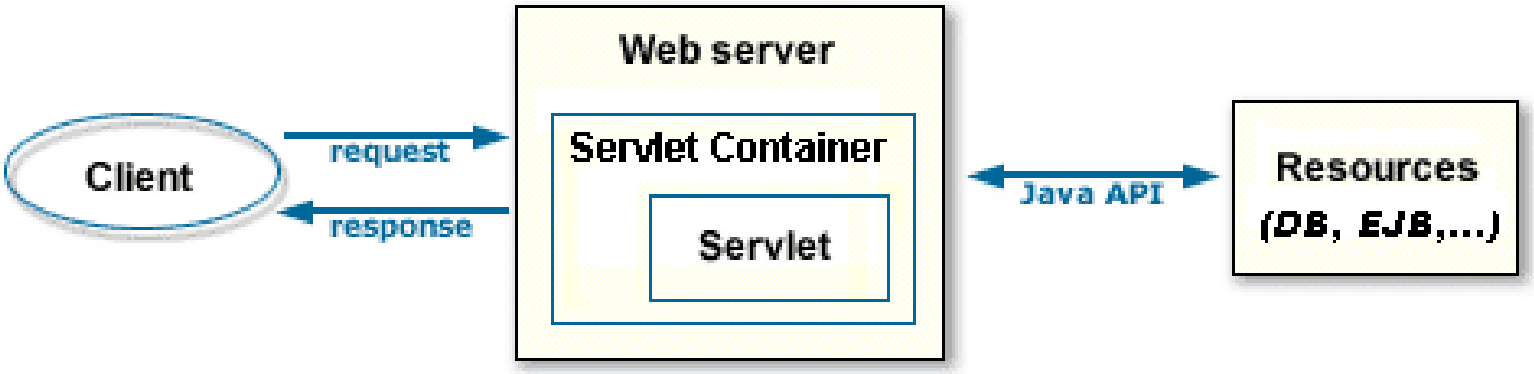
ServletContainer

Il servlet container (o servlet engine) è un’estensione di un web server che fornisce l’ambiente di esecuzione ad una Servlet.

- **Naturalmente le Servlet sono "persistenti" ma non per questo necessariamente "immortali". Hanno un momento**
  - **d'inizializzazione, consentito dalla chiamata del metodo init();**
  - **un'attività vitale gestita dal metodo service() al servizio delle successive chiamate;**
  - **infine, vengono eliminate tramite il metodo destroy().**
- **All'occorrenza del metodo destroy(), le Servlet "muoiono" ma non vengono dimenticate, saranno infatti archiviate in memoria tutte le informazioni necessarie per i successivi processi creativi.**

Come funziona?

- Il Client fa una **richiesta** HTTP al web server.
- Il web server **carica** la servlet (solo la prima volta) e **crea** un thread per eseguirla.
- Il container **esegue** la servlet richiesta.
- La servlet **genera** la risposta.
- La risposta viene **restituita** al client.



Benefici offerti da Servlet

- **Sono molto più veloci dei CGI**
- **Persistenti: una servlet una volta caricata rimane in memoria e può ottimizzare l'accesso alle risorse attraverso caching, pooling, etc…**
- **Implementation independence: usano una API standard supportata da molti web server**
- **Vantaggi offerti dal linguaggio Java (Platform independence, OO, GC…)**

- **Esistono diverse tipologie di servlet, che ereditano da un supertipo presente nel package javax.servlet. La loro estensione dipende dal protocollo di comunicazione utilizzato.**

- **Il modo di funzionare ricalca il paradigma client / server, dove a seguito di una richiesta di servizio, c'è una risposta opportuna. La simulazione di queste due funzioni è garantita dagli oggetti delle classi:**

HttpServletRequest HttpServletResponse

contenuti nel package javax.servlet.http.

- **HttpServletRequest** rappresenta la richiesta effettuata dal client al servizio, quindi alla servlet. Tutte le informazioni necessarie al servizio sono esplicitate dalla presenza di opportuni metodi.
- La servlet può recuperare tutti i parametri utilizzando i metodi dell'oggetto di **HttpServletRequest**, che, mediante il protocollo HTTP li trasporta verso il web server e quindi verso la servlet.
- **HttpServletResponse** rappresenta la risposta che viene inviata al client

Servlet Lifecycle

- **Load & instantiation**

```
Servlet svt = new HttpServlet();
```

- **Initialization**

```
svt.init(ServletConfig);
```

Questo metodo viene chiamato una sola volta, subito dopo essere stata istanziata. È in questo metodo che la servlet istanzia tutte le risorse che utilizzerà poi per gestire le richieste

- **Request Handling**

  svt.service(request, response);

  Questo metodo è incaricato di gestire le richieste effettuate dal client, e ovviamente potrà iniziare il suo lavoro esclusivamente dopo la chiamata del metodo init()

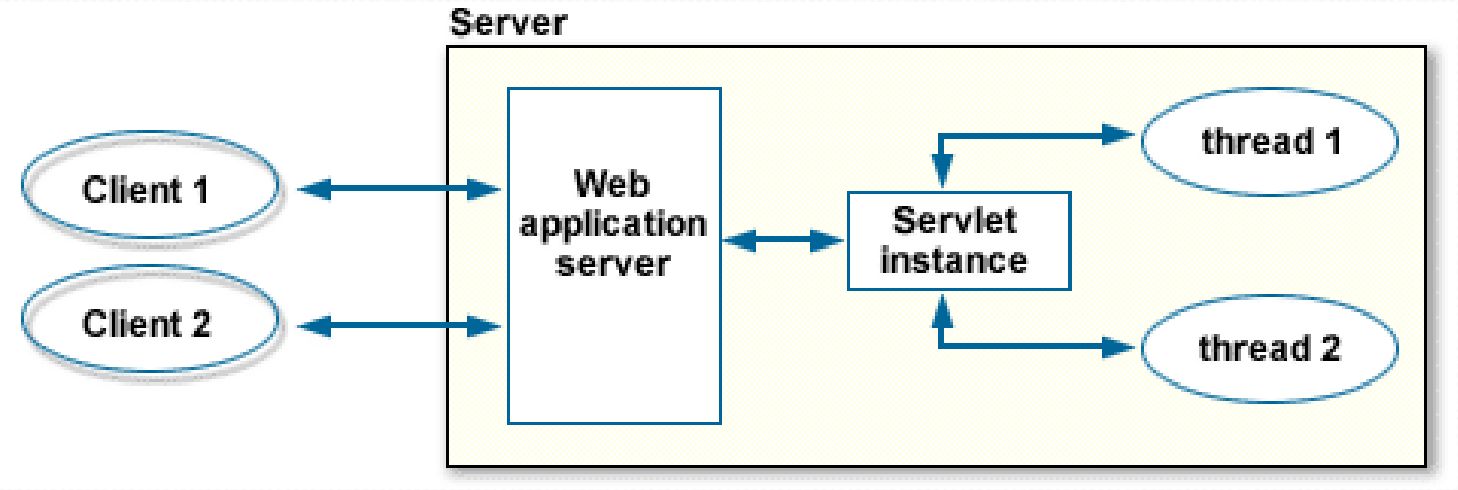
- **End of service**

  svt.destroy();

  Questo metodo segna la chiusura della servlet, è qui che si effettuano eventuali salvataggi di informazioni utili ad un prossimo caricamento

Servlet Lifecycle Multithreading

Viene creato un Thread per ogni richiesta. Il thread può essere riutilizzato se lo stesso client richiede la servlet.



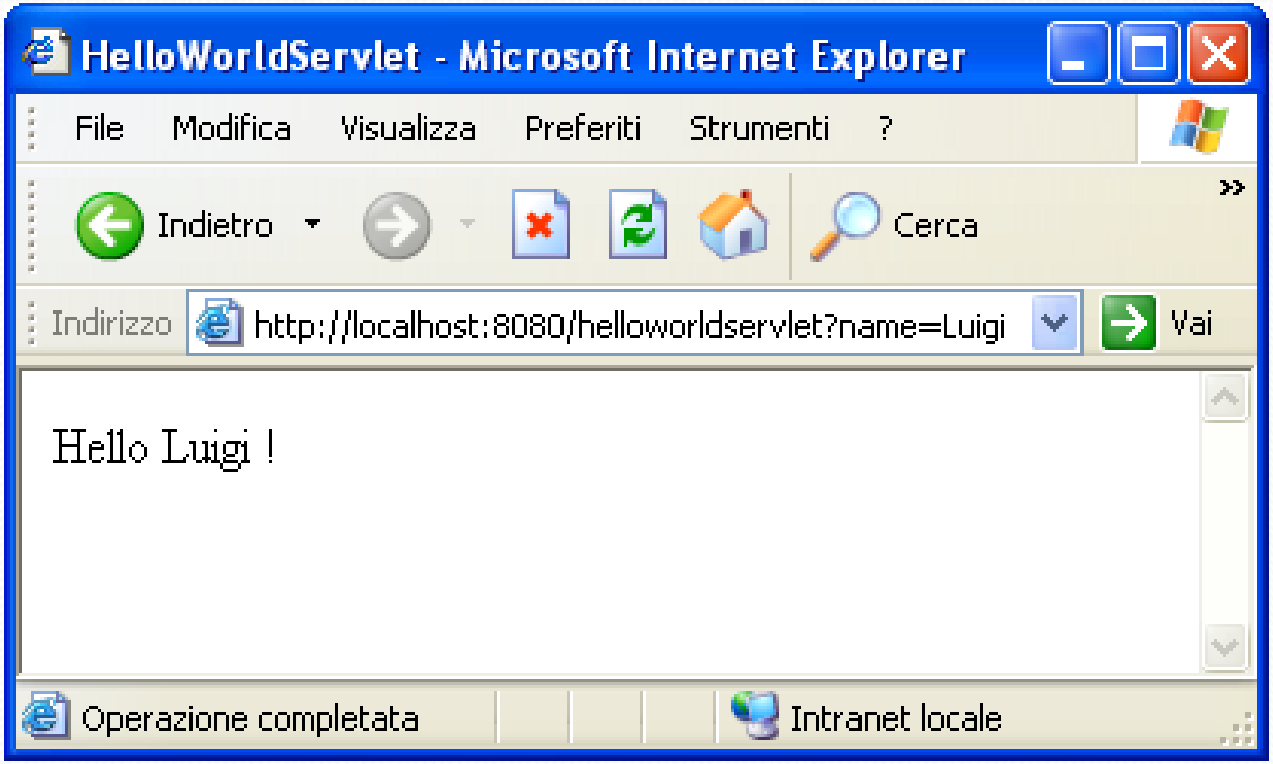
Servlet API

- **E' un framework di classi Java che offre delle interfacce object oriented che incapsulano la comunicazione tra client e server (request, response).**
- **Implementa meccanismi di session tracking per implementare applicazioni complesse.**

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class HelloWorldServlet extends HttpServlet {
    public void init() throws ServletException { }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        if (name == null)
            name = "world";
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>HelloWorldServlet</title></head>");
        out.println("<body>");
        out.println("<p>Hello " + name + " !</p>");
        out.println("</body></html>");
    }

    public void destroy() { }
}
```

Servlet API: Esempio

Ecco il risultato:



Servlet API Request

Permette di

- **Ottenere i parametri inviati dal client**
- **Ottenere il riferimento alla sessione utente**
- **Ottenere il flusso dei dati inviati dal client**
- **Riconoscere l’utente autenticato**

Servlet API Response

Permette di

- **Inviare dati al client sottoforma di HTML oppure in un flusso binario**
- **Inviare codici di errore e codici di controllo nell'intestazione della response HTTP per controllare il comportamento del browser**

Servlet API Session

```
HttpSession session = request.getSession(true);
```

Il protocollo HTTP è un protocollo senza stato (stateless).

Per costruire applicazioni web complesse è necessario riconoscere il client che ci ha inviato la richiesta, e poter suddividere operazioni complesse in più parti, memorizzando uno stato (Sessione).

Servlet API Session

La sessione permette di

- **salvare degli oggetti nella memoria del server che rappresentano lo stato delle operazioni svolte da uno specifico client.**
- **ottenere dei dati memorizzati da servlet eseguite precedentemente.**

Servlet API
init()

- **Il metodo init() viene eseguito una volta soltanto per ciascuna servlet.**
- **Permette di accedere a risorse utili per la servlet (aprire connessioni al db, ottenere reference ad EJB, etc…)**

Servlet API
service/doGet/doPost

- **I metodi service, doGet() e doPost() possono essere implementati dal programmatore.**
- **Il metodo service() per default delega l'esecuzione al metodo indicato nella richiesta HTTP del client.**
- **Le richieste HTTP più comuni sono POST e GET, gestite dai metodi della servlet doPost() e doGet() implementate dal programmatore.**

Richieste HTTP POST & GET

- La richiesta HTTP POST viene effettuata quando il client deve inviare dei dati al server.

Per esempio fare l'upload di un file, o inviare un testo molto lungo. Eventuali parametri vengono codificati nel "corpo" del messaggio.

- La richiesta HTTP GET viene effettuata quando il client chiede di ottenere dei dati dal server.

Nella richiesta GET possono essere codificati dei parametri.

Richieste HTTP POST vs GET

POST

- I parametri non compaiono nell'indirizzo del browser
- La lunghezza dei parametri è illimitata, perché i parametri sono codificati nel corpo della richiesta HTTP

GET

- I parametri sono specificati e compaiono nell'indirizzo del browser
- La lunghezza dei parametri è limitata (tipicamente 255 caratteri)

GET : browser richiede semplicemente il contenuto di un particolare URL
POST : browser "invia" dati a un particolare URL. Ad esempio, può inviare i dati di un utente ha inserito in un modulo su una pagina web.

In pratica, vi è una certa sovrapposizione tra i due metodi.
Questo perché sia GET che POST possono effettivamente essere usati per inviare i dati da un modulo web.
Il GET è adatto per i dati di breve durata e / o casi in cui è opportuno che i parametri per essere 'visibili' nella URL.
Con il POST, invece, qualsiasi impostazione del parametro è "nascosta" con la comunicazione con il server.

dex.jsp | web.xml | HelloWorld.java

```java
 * Servlet implementation class HelloWorld
 */
@WebServlet("/HelloWorld")
public class HelloWorld extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public HelloWorld() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
           // reading the user input
        String color= request.getParameter("color");

        PrintWriter out = response.getWriter();
        response.setContentType("text/html");
        out.println("<font size=\"12px\" color=\"" + color + "\">" +
                "Hello World" +
              "</font> \n");

    }
```

dex.jsp | web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://jav
id="WebApp_ID" version="3.0">

  <display-name>EsercizioCambiaColore</display-name>

  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/HelloServlet</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

index.jsp

```
<%@ page language="java" contentType="text/html; charset=I
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional/
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>Cambia Colore scritta</title>
</head>
<body>
    <form action="HelloServlet">
         Please enter a color <br>
        <input type="text" name="color"size="20px">
        <input type="submit" value="submit">
    </form>
</body>
</html>
```

http://localhost:8080/EsercizioCambiaColore/

Pagina Sicurezza Strumenti

Please enter a color

submit

http://localhost:8080/EsercizioCambiaColore/HelloServlet?color=red

Pagina Sicurezza Strumenti

Hello World

Esempio

Creare una pagina web (index.isp) che permetta di inserire Cognome, nome, età e sesso di una persona

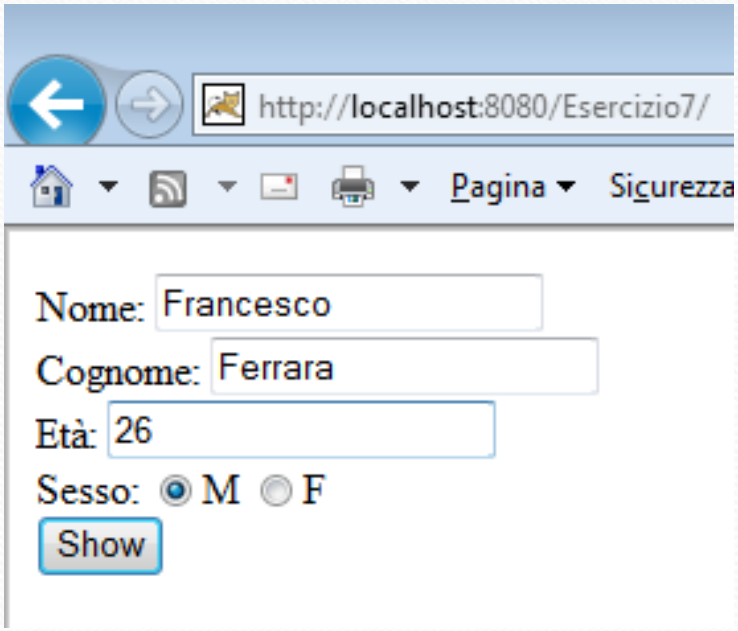
Stampare :

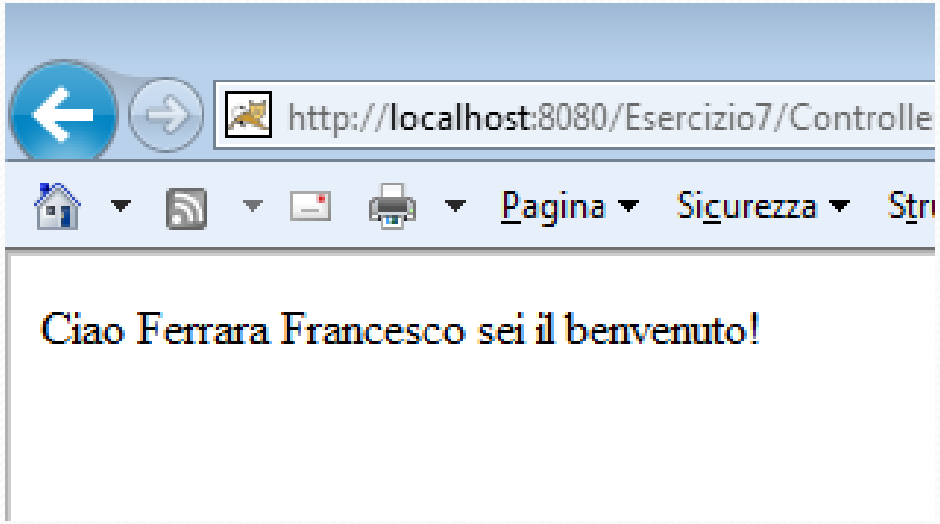
Ciao <u>cognome nome</u> sei la benvenuta

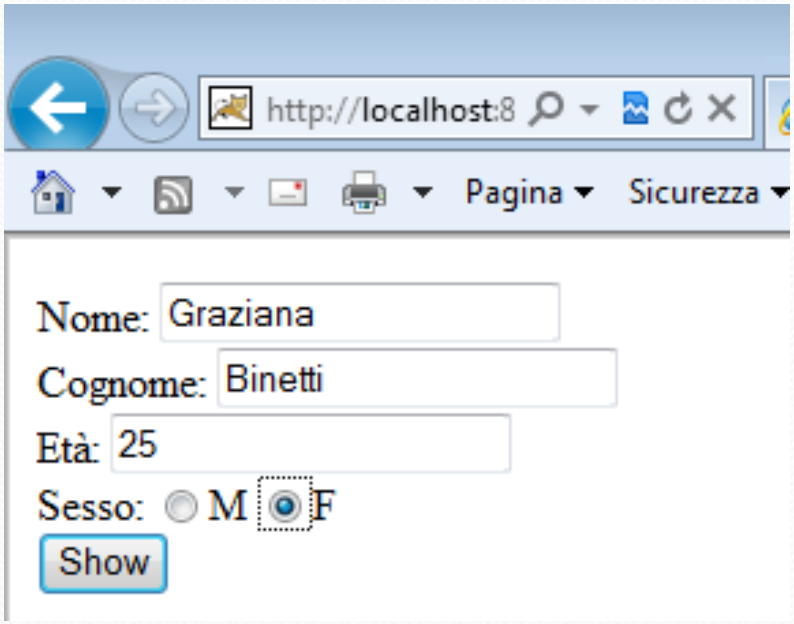
nel caso in cui il sesso ha valore F

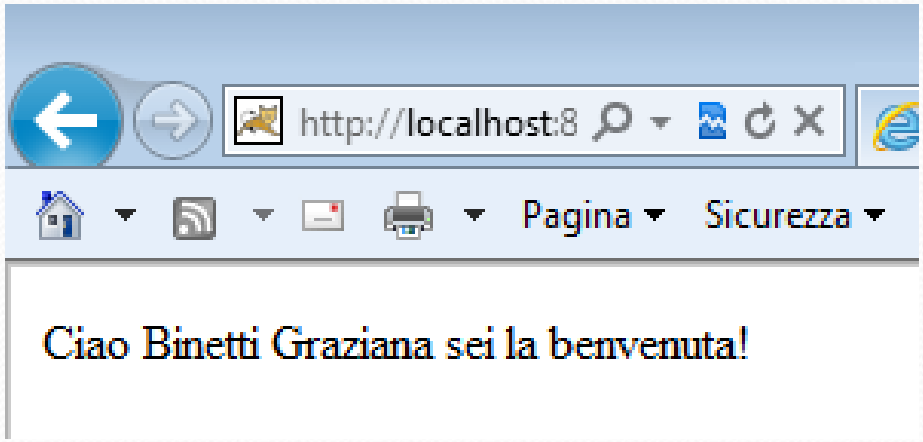
Ciao <u>cognome nome</u> sei il benvenuto

nel caso in cui il sesso ha valore M


http://localhost:8080/Esercizio7/
Pagina
Sicurezza
Nome: Francesco
Cognome: Ferrara
Età: 26
Sesso: M F
Show


http://localhost:8080/Esercizio7/Controlle
Pagina
Sicurezza
Ciao Ferrara Francesco sei il benvenuto!


http://localhost:8
Pagina
Sicurezza
Nome: Graziana
Cognome: Binetti
Età: 25
Sesso: M F
Show


http://localhost:8
Pagina
Sicurezza
Ciao Binetti Graziana sei la benvenuta!

*index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.or
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

      <form name="Login" action='Controller'>
        Nome: <input name="nome" type="text" /><br />
        Cognome: <input name="cognome" type="text" /><br />
        Età: <input name="età" type="text" /><br />
        Sesso: <input name="sesso" type="radio" value="M"/>M
                <input name="sesso" type="radio" value="F"/>F<br />
               <input type="submit" value="Show" />
      </form>

</body>
</html>
```

index.jsp | web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XML
  <display-name>Esercizio7</display-name>
    <servlet>
    <servlet-name>Controller</servlet-name>
    <servlet-class>Controller</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Controller</servlet-name>
    <url-pattern>/Controller</url-pattern>
   </servlet-mapping>
  <welcome-file-list>

    <welcome-file>index.jsp</welcome-file>

  </welcome-file-list>
</web-app>
```

```java
@WebServlet("/Controller")
public class Controller extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        // TODO Auto-generated method stub
        String sesso = request.getParameter("sesso");
        PrintWriter out=response.getWriter();
        out.print("Ciao ");
        out.print(request.getParameter("cognome")+" " +request.getParameter("nome"));
        if ( sesso.equals("M") )
        {
          out.println(" sei il benvenuto! ");
        }
        else if ( sesso.equals("F") )
        {
            out.println(" sei la benvenuta! ");
        }
```

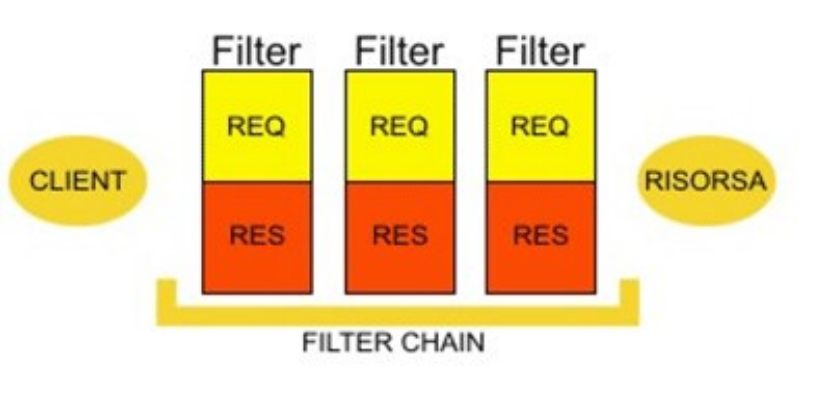
I filtri

Un filtro è una classe in grado di intercettare una richiesta e eseguire elaborazioni sulla richiesta stessa o sulla risposta generata da una servlet.

- **Queste funzionalità possono essere aggiunte o eliminate in maniera indipendente l'una dall'altra e con una semplice configurazione esterna al codice**

- **I filtri vengono configurati nel *deployment descriptor* dper intercettare tutte le richieste del client ad una determinata servlet o a un gruppo di servlet individuate da uno specifico URL pattern**

- **L'elaborazione della richiesta da parte di un filtro avviene prima che questa arrivi alla servlet a cui era destinata la richiesta.**



I filtri

Una classe per essere un filtro deve implementare l'interfaccia standard javax.servlet.Filter che prevede tre metodi:

- ***public void init()*** **È il metodo invocato dal web container per inizializzare il filtro**

- ***public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)*** **È il metodo invocato dal web container per eseguire l'elaborazione del filtro quando questo è invocato nella catena di filtri**

- ***public void destroy()*** **È il metodo invocato dal web container per mettere fuori servizio il filtro**

```
public  class FilterSample implements Filter{
    private FilterConfig filterConfig = null;
    public void init (FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
    }
    public void destroy () {
        this.filterConfig = null;
    }
    public void doFilter (ServletRequest request, ServletResponse response, FilterChain chain)
    throws IOException, ServletException {
        //Elaborazione REQUEST…
        chain.doFilter(request, resonse);
        //Elaborazione RESPONSE…
    }
}
```

I filtri

- **La determinazione della sequenza dei filtri della catena è fatta nel *deployment descriptor* dell'applicazione. Definire un filtro è molto simile alla definizione di una servlet**
- **I filtri sono particolarmente adatti per operazioni quali:**
  - **Autenticazione**
  - **Logging e Auditing**
  - **Conversione di immag**
  - **Compressione dati**
  - **Encryption**
  - **Tokenizing**
  - **Trasformazioni XSLT**

```
<filter>
    <filter-name>LoggingFilter</filter-name>
    <filter-class>it.mokabyte..LoggingFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoggingFilter</filter-name>
    <url-pattern*.do</url-pattern>
</filter-mapping>
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Esempio

- **Vogliamo realizzare un'applicazione che contenga due tipi di pagine:**
  - **Pagine pubbliche, visibili a tutti**
  - **Pagine riservate, visibili solo agli utenti registrati**
- **Per accedere alle pagine riservate l'utente deve fornire login e password**
- **Le pagine riservate possono essere anche pagine statiche (HTML),**
- **Non deve essere possibile accedere alle pagine riservate scavalcando la verifica di login e password, neanche conoscendo l'URL di una pagina riservata**

- **Soluzione:**
- **inseriamo tutte le pagine riservate in una sottocartella riservate**
- **definiamo un filtro che controlla gli accessi al contenuto di riservate**
- **il filtro userà informazioni di sessione per verificare se l’utente è stato autenticato**
- **la verifica di login e password viene effettuata da una servlet associata a un form**

Home page

Questa è una pagina pubblica.
Clicca qui per vedere una pagina riservata. Clicca qui per vedere un'altra pagina riservata.

Inserimento login e password

Per accedere alla sezione riservata devi fornire login e password.

Login:

Password:

Invia

Inserimento login e password

Per accedere alla sezione riservata devi fornire login e password.

Login: pippo

Password: •••••

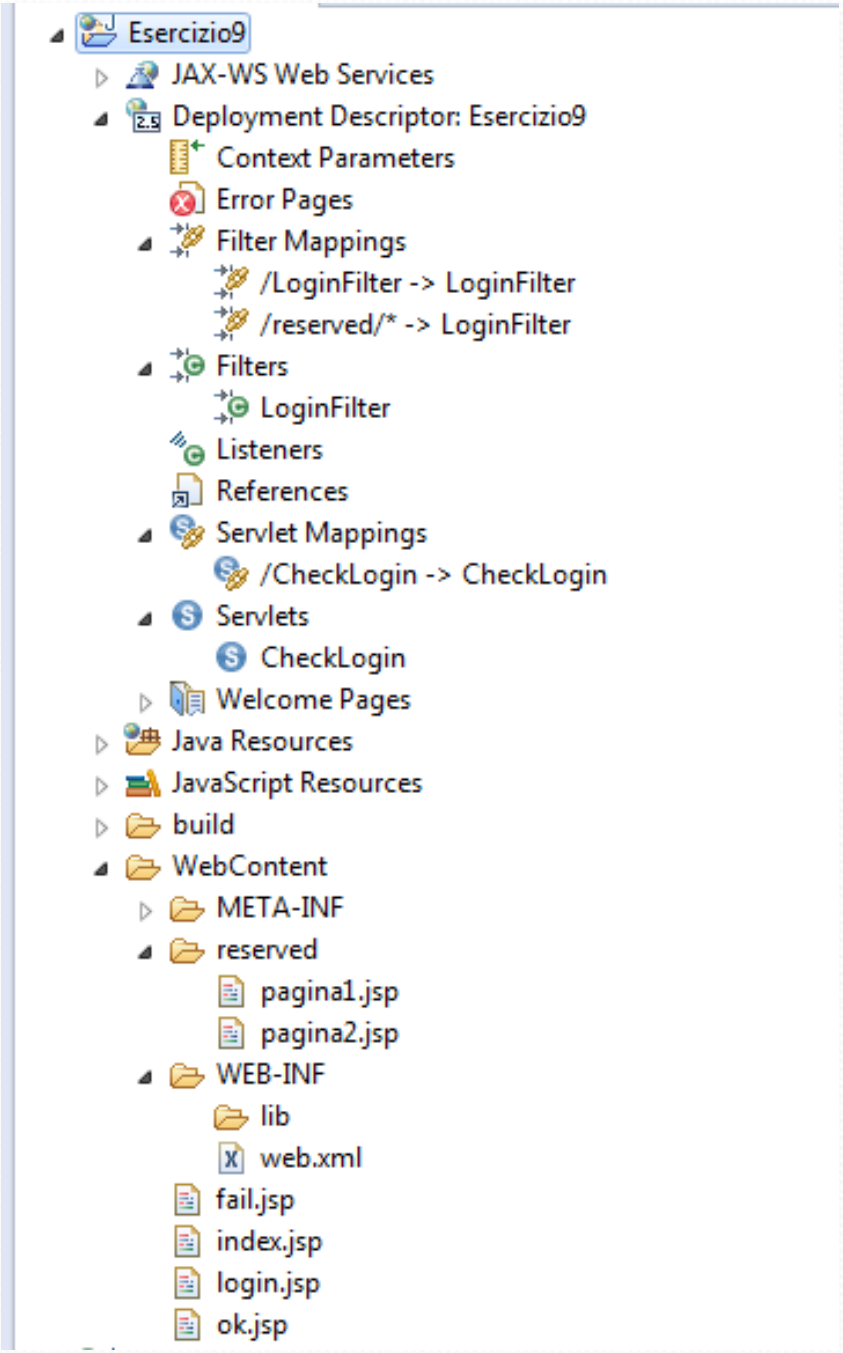
Invia

Autenticazione eseguita

Login e password verificati.
Clicca qui per tornare alla home page e accedere alle pagine riservate.

Questa è una pagina riservata!

Questa è la pagina riservata 2.
Clicca qui per tornare alla home page.
Clicca qui per vedere la prima pagina riservata.


Esercizio9
JAX-WS Web Services
Deployment Descriptor: Esercizio9
Context Parameters
Error Pages
Filter Mappings
/LoginFilter -> LoginFilter
/reserved/* -> LoginFilter
Filters
LoginFilter
Listeners
References
Servlet Mappings
/CheckLogin -> CheckLogin
Servlets
CheckLogin
Welcome Pages
Java Resources
JavaScript Resources
build
WebContent
META-INF
reserved
pagina1.jsp
pagina2.jsp
WEB-INF
lib
web.xml
fail.jsp
index.jsp
login.jsp
ok.jsp

index.jsp | LoginFilter.java | web.xml | CheckLogin.java

```
 */
@WebFilter("/LoginFilter")
public class LoginFilter implements Filter {

    private ServletContext ctx;

    public void init(FilterConfig cfg) throws ServletException {
    ctx = cfg.getServletContext();
    }
```

index.jsp | *LoginFilter.java | web.xml | CheckLogin.java

```
public voi[Esercizio9/src/LoginFilter.java]equest request, ServletResponse response, FilterChain c
            throws IOException, ServletException {
        // TODO Auto-generated method stub
        // place your code here

    HttpServletRequest httpRequest = (HttpServletRequest)request;
    HttpServletResponse httpResponse =(HttpServletResponse)response;
    HttpSession session = httpRequest.getSession();
    String auth =
    (String)session.getAttribute("authenticated");
    if ("yes".equals(auth))
            chain.doFilter(request, response);
    else
            httpResponse.sendRedirect(ctx.getContextPath() + "/login.jsp");

    }
```

index.jsp | LoginFilter.java | web.xml | CheckLogin.java

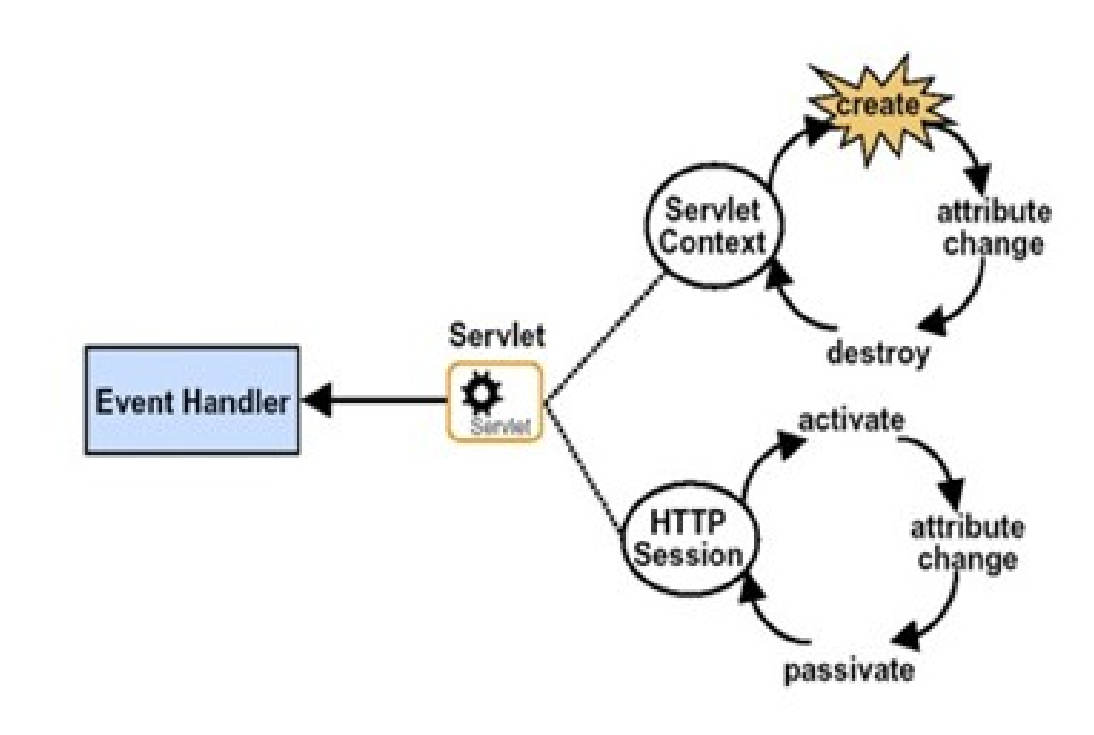
```java
    protected void doPost(HttpServletRequest request, HttpServletResponse
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        HttpSession session=request.getSession();

        String trueLogin = getInitParameter("login");
        String truePass = getInitParameter("pass");

        String login = request.getParameter("login");
        String pass = request.getParameter("pass");
        RequestDispatcher disp;

        if (trueLogin.equals(login) &&
                truePass.equals(pass)) {
                session.setAttribute("authenticated", "yes");
                disp = request.getRequestDispatcher("ok.jsp");
                disp.forward(request, response);
        } else {
                session.setAttribute("authenticated", "no");
                disp = request.getRequestDispatcher("fail.jsp");
                disp.forward(request, response);
                }
    }
```

index.jsp | LoginFilter.java | *web.xml | CheckLogin.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSche
  <display-name>Esercizio8Filtro</display-name>
  <servlet>
    <servlet-name>CheckLogin</servlet-name>
    <servlet-class>CheckLogin</servlet-class>
        <init-param>
            <param-name>login</param-name>
            <param-value>pippo</param-value>
        </init-param>
        <init-param>
            <param-name>pass</param-name>
            <param-value>baudo</param-value>
        </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>CheckLogin</servlet-name>
    <url-pattern>/CheckLogin</url-pattern>
</servlet-mapping>
<filter>
    <filter-name>LoginFilter</filter-name>
    <filter-class>LoginFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>LoginFilter</filter-name>
    <url-pattern>/reserved/*</url-pattern>
</filter-mapping>
```

I Listener

- **Un listener permette di registrare gli eventi che si verificano all'interno di una web application e nel corso dell'interazione con l'utente.**



I Listener

- **Le interfacce definite nella Servlet API che un listener deve implementare sono le seguenti:**
  - **javax.servlet.ServletContextListener**
  - **javax.servlet.ServletContextAttributeListener**
  - **javax.servlet.http.HttpSessionListener**
  - **javax.servlet.http.HttpSessionActivationListener**
  - **jvax.servlet.http.HttpSessionAttributeListener**
  - **javax.servlet.ServletRequestListener**
  - **javax.servlet.ServletRequestAttributeListener**

I Listener

- **I listener vengono configurati nel web.xml dell'applicazione aggiungendo una sezione come nell'esempio seguente:**

```
<listener>
    <listener-class>
        MioListener
    </listener-class>

</listener>
```

- **Non è necessario specificare il tipo del listener perché il container lo desume ispezionando la classe registrata nel web.xml e vedendo quale interfaccia implementa.**

- **I listener vengono invocati nell'ordine in cui sono definiti nel web.xml tranne i listener per gli eventi di shutdown (distruzione del contesto) che sono invocati in ordine inverso.**

Custom Tag

Un Custom Tag si presenta come un componente XML da usare all'interno delle pagine jsp, che verrà interpretato dal web container e trasformato opportunamente in codice html.

I principali vantaggi nell'usare questo componente sono:

- **Riutilizzo all'interno di diversi punti dell'applicazione;**
- **Utilizzo dei tag come estensione di (X)HTML;**
- **Incapsulamento di logiche applicative in singole unità di sviluppo;**
- **Eleganza e chiarezza nella produzione delle pagine JSP.**

Custom Tag

- **Un Custom Tag viene utilizzato all'interno di una pagina JSP come tag XML "personalizzato" diventando un componente dell'applicazione.**
- **Per poter essere interpretato correttamente sono necessari**
  - **il descrittore del tag TLD (Tag Library Description)**
  - **un documento XML che regola l'uso dei tag all'interno delle pagine JSP**
  - **lo sviluppo di una o più classi Java (tag handlers) che definiscono la logica che risponde al tag**
- **Quello che viene fuori, dinamicamente sarà quindi codice HTML che sarà interpretato dal browser.**

Architettura client-server

- **Compiti del browser**
  - **visualizzare le risorse e gestire l'interazione con l'utente**
- **Compiti del server**
  - **nel caso di documenti: inviare il contenuto di file contenuti sul disco locale**
  - **nel caso di servizi interattivi: eseguire un'applicazione e restituire i risultati dell'esecuzione**

Server Web

- **Applicazione software**
  - **che fornisce accesso ad un sito Web**
- **Caratteristiche**
  - **fornisce servizi basati su HTTP**
  - **consente l’accesso a documenti statici**
  - **consente l’accesso a servizi interattivi (applicazioni) >> server applicativo**

Server Web

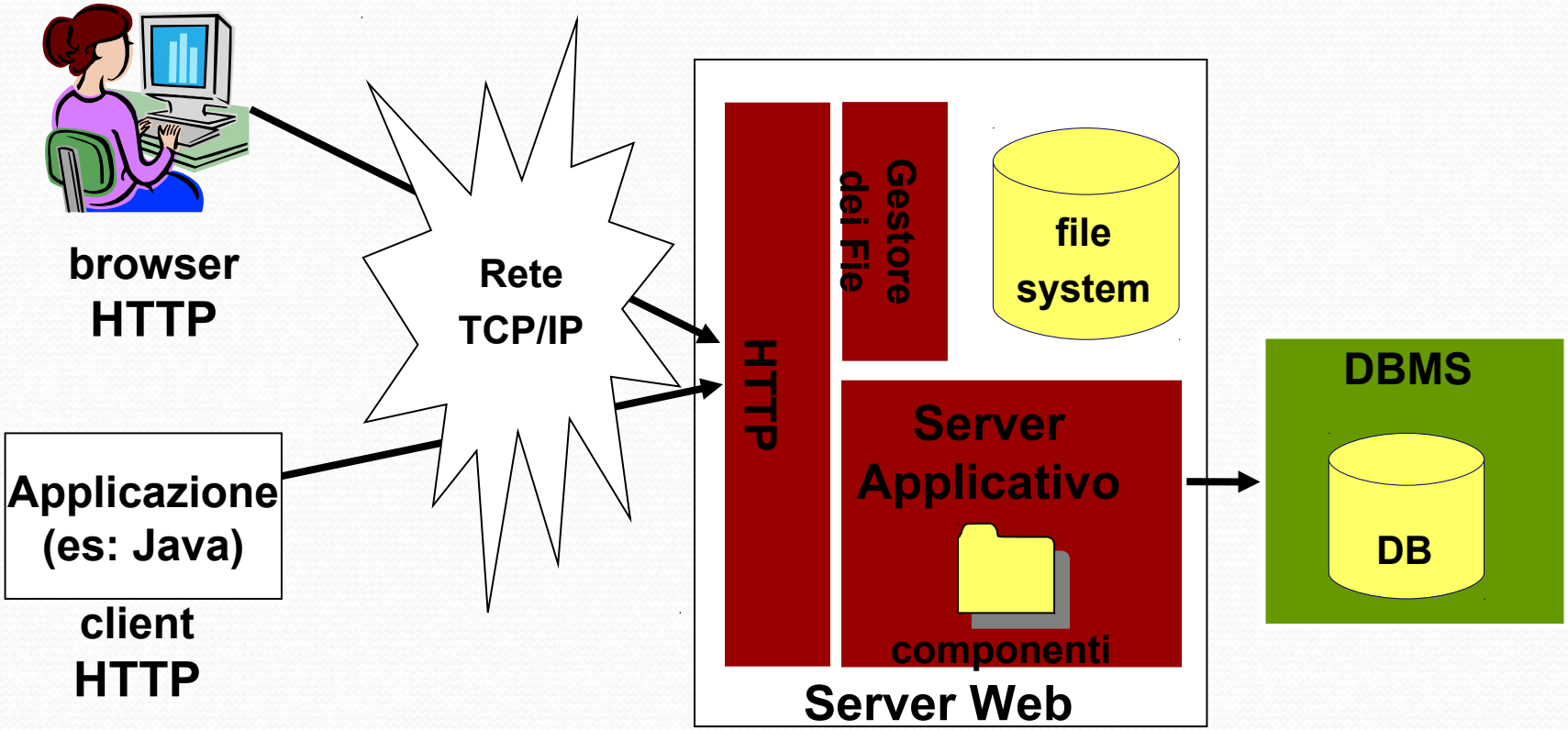
- **Apache HTTP Server (httpd.apache.org)**
  - **open source (deriva da NCSA)**
  - **server HTTP**
  - **vari protocolli di interfaccia con server applicativi (es: CGI, PHP)**
- **Microsoft Internet Information Services (IIS)**
  - **server HTTP**
  - **integrato con il server applicativo .NET**

Server Web

- **Apache Tomcat (jakarta.apache.org)**
  - **server applicativo open source per Servlet e JSP**
  - **include server HTTP**
- **IBM WebSphere, Bea WebLogic ed altri**
  - **server applicativi commerciali per J2EE**
  - **includono server HTTP**
- **JBoss (www.jboss.org)**
  - **server applicativo open source per J2EE**
  - **include server HTTP**

Architettura di un Server Web

- Contiene vari moduli
- Server HTTP
  - implementa il protocollo HTTP
  - include vari altri servizi; es: caching, logging
- Gestore del file system
  - contenuti statici salvati come file
- Server applicativo
  - gestore di applicazioni e componenti

Architettura di un Server Web

- **Architettura a 3 livelli**