AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Applicazioni web

Parte 2

JSP

Michele Tomaiuolo

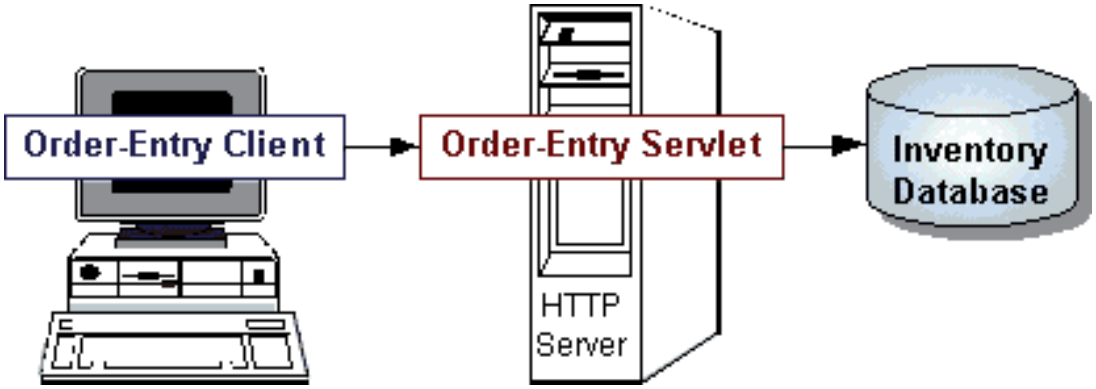
tomamic@ce.unipr.it

Sommario

- JSP: sintassi di base
- JavaBeans: componenti Java, uso in pagine JSP
- EL: linguaggio per espressioni
- JSTL: libreria di tag, azioni principali, azioni per SQL
- Servlet: introduzione alle API ed esempi
- Architettura di applicazioni web complesse
- Installazione su Tomcat

Servlet Introduzione

- Le servlet sono moduli che estendono server di tipo richiesta/risposta, come i server web con supporto Java



  - Una servlet può occuparsi di accettare i dati di un form html relativi a un nuovo ordine e aggiornare un database applicando la logica aziendale

- Le servlet possono essere incluse in diversi tipi di server
  - Le API delle servlet non fanno nessuna assunzione sull'ambiente o il protocollo del server

JSP
Java Server Pages

- Le specifiche JSP sono una estensione standard definita al di sopra delle API per le servlet
  - Si può sfruttare l'esperienza sulle servlet
  - Sviluppo dichiarativo, centrato sulla presentazione
  - Per disegnatori HTML, non solo per programmatori Java
- Le pagine JSP di solito consistono di:
  - Elementi statici HTML
  - Commenti e direttive
  - Espressioni (EL) e tag speciali JSP (azioni)
  - Frammenti di codice java denominati "scriptlets" (sconsigliati!)
- Si possono creare con i comuni editor di testo

- *http://java.sun.com/products/jsp/*

JSP
Hello, user!

```
<html>
<head>
  <title>Hello user</title>
</head>
<body>

<i>Hello, ${param.user}!</i><br/>

</body>
</html>
```

JSP
Commenti e direttive

- Racchiudere i commenti in tag `<%-- ... --%>`
  - `<!-- commento inviato al browser, nel codice HTML -->`
  - `<%-- commento solo per la parte server --%>`

- Le direttive sono racchiuse in tag `<%@ ... %>`
- Sono messaggi per l'engine jsp
  - Non producono direttamente un output visibile
  - In fase di traduzione, indicano cosa fare col resto della pagina
- Le direttive principali sono: *page*, *include*, *taglib*

JSP
Direttive page e include

- A inizio pagina, numero arbitrario di direttive *page*
  - La coppia attributo/valore deve essere unica
  - Attributi o valori non riconosciuti generano errori nella traduzione
- Es. package resi disponibili per scriptlet, 16KB come buffer
  - `<%@ page import="java.util.*, com.foo.*" buffer="16k" %>`

- Direttiva *include* per dividere l'elaborazione tra più file
  - Includere in più pagine uno stesso header o footer
  - La pagina inclusa può essere html statico o altro jsp
- La risorsa viene inclusa a tempo di traduzione
- Esempio
  - `<%@ include file="common-header.html" %>`

JSP Azioni

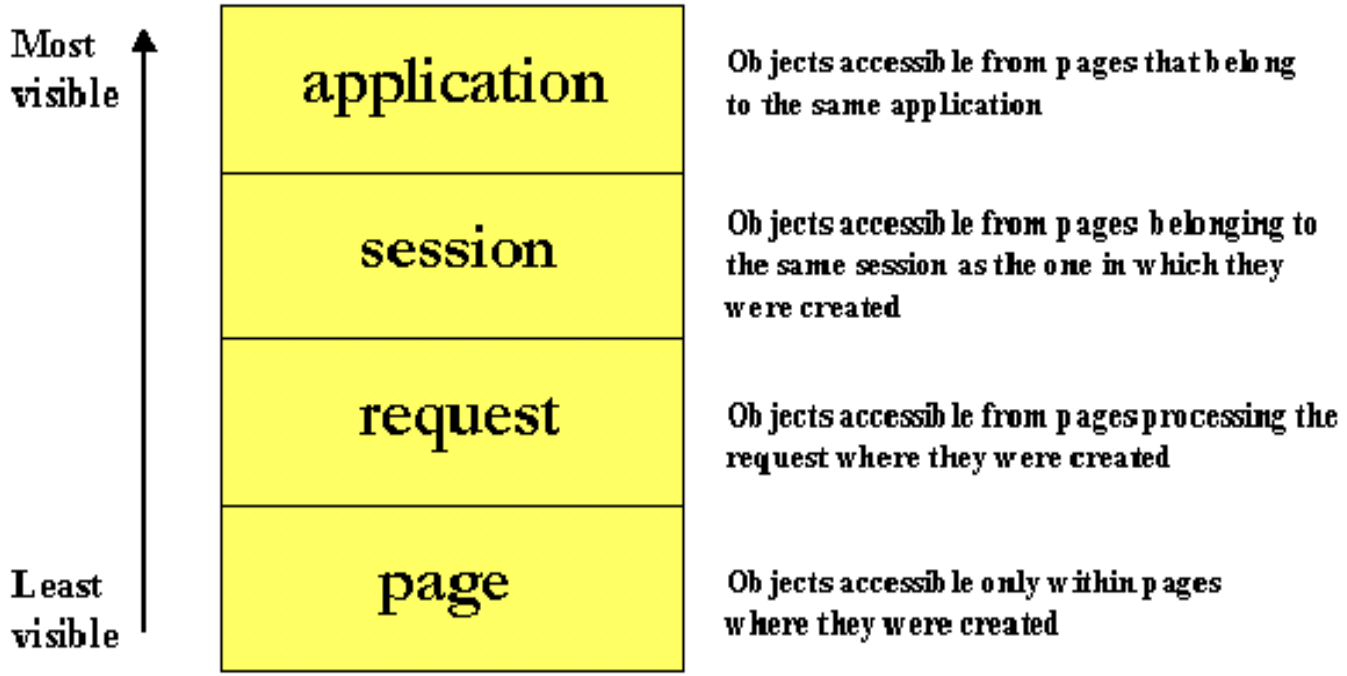
- Eseguite durante la fase di gestione delle richieste
  - Es. per creare e manipolare componenti
  - Azioni *standard* (definite in specifiche JSP, con prefisso “jsp”)…
  - Oppure *custom* (fornite tramite meccanismi di estensione)

- Una “azione” segue la sintassi degli elementi xml
  1. Tag di apertura, attributi e corpo opzionali, tag di chiusura
  2. Oppure un tag semplice, con eventuali attributi

  - `<mytag attr1="value" ... >body</mytag>`
  - `<mytag attr1="value" ... />`
  - `<mytag attr1="value" ... ></mytag>`

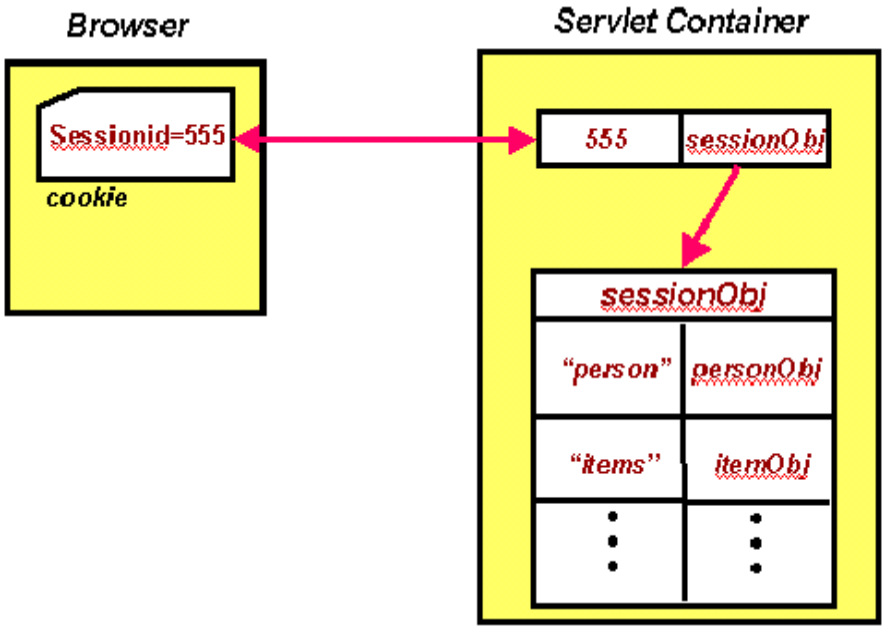
JSP
Visibilità degli oggetti

- Le pagine jsp possono gestire vari tipi di oggetti
  - Oggetti impliciti, creati automaticamente dal container jsp
  - Componenti, creati dagli autori della pagina



JSP
Gestione delle sessioni

- Per default, tutte le pagine JSP partecipano ad una sessione HTTP
- Utile per memorizzare oggetti condivisi tra varie pagine e servlet cui l'utente ha accesso
- Si può accedere alla sessione tramite un oggetto implicito di classe *HttpSession*
- Oggetto sessione identificato da un *session-id*, memorizzato come cookie sul browser



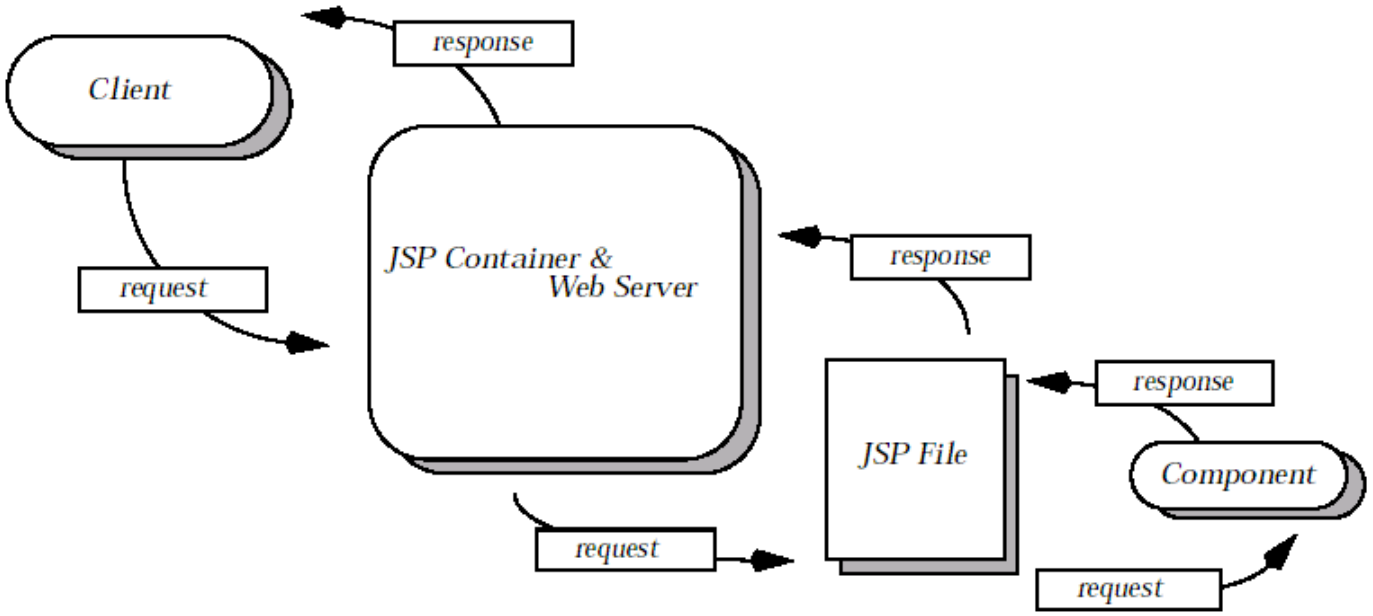
JSP
Contesto di sessione

- Si può memorizzare in una sessione qualsiasi oggetto java
- Deve essere identificato da una chiave univoca
- No limiti al numero di oggetti memorizzati in una sessione
  - Tuttavia, porre grossi oggetti nella sessione può degradare le prestazioni, consumando prezioso spazio in memoria heap

- Una JSP non deve per forza partecipare a una sessione
  - Attributo appropriato in direttiva page
  - `<%@ page session="false" %>`
- I server impostano il tempo di vita di un oggetto sessione
  - Spesso nei server il default è 30 minuti
  - Si può modificare invocando il metodo *setMaxInactiveInterval(int secs)* sull'oggetto *session*

JavaBeans
Elaborazione e presentazione

- La sola funzione delle pagine JSP dovrebbe essere di *presentare* contenuti dinamici ai loro utenti
- Reale *elaborazione* dati, fuori dal codice JSP
  - Disegnatori di pagine non necessariamente abili sviluppatori



JavaBeans
Componenti Java

- I *JavaBeans* sono *componenti* software in linguaggio java, sviluppati come semplici oggetti
- I componenti sono unità software **auto-contenute e riutilizzabili**
- … che possono essere **composte** in applicazioni, servlet e altri componenti
- … anche usando strumenti di sviluppo **visuali**

- Le proprietà sono valori privati
  - Riguardano aspetto e comportamento di un bean, modificabili sia in fase di sviluppo che esecuzione
  - Vi si accede tramite metodi *getter* e *setter*, i cui nomi seguono specifiche regole, chiamate design pattern

JavaBeans
Metodi getter e setter

```
public class Person implements java.io.Serializable {
  public Person() { // constructor
  }

  private String name = null;
  public String getName() { // property getter method
    return name;
  }
  public void setName(String name) { // property setter method
    this.name = name;
  }

  private int age = 0;
  public int getAge() { // property getter method
    return age;
  }
  public void setAge(int age) { // property setter method
    this.age = age;
  }
}
```

JavaBeans
Azione useBean

- Prima di poter accedere a un bean da una pagina JSP, è necessario identificarlo e ottenerne un riferimento!
- L'azione *useBean* tenta di recuperare un riferimento ad una istanza esistente usando lo *id* e lo *scope* specificati
  - Il bean può essere stato creato precedentemente e posto nello scope sessione o app. nell'esecuzione di una diversa richiesta
- Il bean è istanziato nuovamente (in accordo all'attributo *class*) solo se un riferimento non viene trovato
- Si consideri il tag
  - `<jsp:useBean id="user" class="somepackage.Person" scope="session" />`
  - Istanza di *Person* creata una sola volta e memorizzata in sessione
  - Se tag incontrato nuovamente, in una nuova richiesta dello stesso utente, recuperato da sessione un riferimento alla istanza già creata

JavaBeans
Azione setProperty

- Modificare una proprietà di un componente JavaBean richiede l'uso del tag *setProperty*
- Per questo tag, serve identificare il bean, la proprietà da modificare e il nuovo valore da memorizzare:
  - `<jsp:setProperty id="user" property="name" value="tomamic" />`
  - `<jsp:setProperty id="user" property="name" param="username" />`
- Il tag *useBean* opzionalmente può anche includere un corpo, che viene eseguito solo quando il bean viene istanziato

```
<jsp:useBean id="user" class="somepackage.Person"
scope="session">
  <jsp:setProperty id="user" property="age" value="19" />
</jsp:useBean>
```

JavaBeans
Azione getProperty

- Si può accedere al valore di una proprietà di un bean usando il tag *getProperty*
- Si specifica il nome del bean da usare (dal campo *id* del tag *useBean*) assieme al nome della *proprietà* che si vuole leggere
- Il valore viene visualizzato direttamente in output
  - `<jsp:getProperty id="user" property="name" />`

EL
Linguaggio per espressioni

- Caratteristica chiave delle pagine jsp 2.0 è il supporto per un *linguaggio per espressioni*
  - Accedere e manipolare facilmente i dati della applicazione
  - Senza dover usare scriplet o codice java
- EL invocato esclusivamente attraverso il costrutto *${expr}*
- Un *identificatore* in EL fa riferimento a variabili JSP
  - Uno dei quattro livelli di visibilità (scope): *pagina*, *richiesta*, *sessione*, o *applicazione*
- I parametri di richiesta, le intestazioni e i cookie sono accessibili tramite oggetti impliciti: *param*, *header* e *cookie*
  - *param["foo"]* (o *param.foo*) restituisce come stringa il valore associato al parametro di richiesta *foo*

EL
Componenti e collezioni

- I dati dell'applicazione di solito consistono di oggetti che aderiscono alle specifiche dei *JavaBeans*, o che rappresentano *collezioni* come liste, mappe o array
  - EL fornisce due *operatori*, *"."* e *"[]"*, per rendere facile l'accesso ai dati incapsulati in questi oggetti
  - EL segue ECMAScript nell'unificare il trattamento di *"."* e *"[]"*
  - L'operatore *"."* può essere usato come conveniente abbreviazione per accedere alle proprietà, quando il loro nome segue le convenzioni degli identificatori java
  - L'operatore *"[]"* permette un accesso più generale

EL Operatori

- Disponibili soliti operatori relazionali, aritmetici e logici
  - `==` o *eq*, `!=` o *ne*, `<` o *lt*, `>` o *gt*, `<=` o *le*, `>=` o *ge*
  - `+`, `-`, `*`, `/` o *div*, `%` o *mod*
  - `&&` o *and*, `||` o *or*, `!` o *not*
  - Operatore condizionale *? – A ? B : C*
  - Fornito inoltre l'operatore *empty*

- Es.

```
<%@ taglib uri=http://java.sun.com/jsp/jstl/core
    prefix="c" %> [...]
<c:if test="${book.price <= user.spendingLimit}">
  The book ${book.title} fits your
  budget!
</c:if>
```

EL
Oggetti impliciti

- Il linguaggio per espressioni JSP definisce un insieme di oggetti impliciti
- *pageContext*: il contesto della pagina jsp, che fornisce accesso a vari oggetti, tra cui:
  - *servletContext*: il contesto della pagina e di tutti i componenti della stessa *applicazione*
  - *session*: l'oggetto sessione per il client
  - *request*: la richiesta che attiva l'esecuzione della pagina jsp
  - *response*: la risposta restituita dalla pagina jsp

EL
Oggetti impliciti

- Diversi altri oggetti impliciti sono disponibili:
  - *param*: mappa un nome di parametro di richiesta al suo valore
  - *cookie*: mappa un nome di cookie al suo valore
  - *header*: mappa un nome di header della richiesta al suo valore
  - *initParam*: mappa un nome di parametro di inizializzazione al suo valore
  - *paramValues, headerValues*: per valori multipli, restituiscono array (es. *select* con attributo *multiple*)

JSTL
Usare una libreria di tag

- L'insieme di tag significativi che un container JSP può interpretare può essere esteso attraverso una *tag library*
  - Collezione di azioni da usare in una pagina JSP
- La direttiva *taglib* dichiara che la pagina usa una tag library
  - Identifica univocamente la tag library tramite una uri
  - Associa un prefisso di tag per distinguere l'uso delle azioni della libreria
  - `<%@ taglib uri="..." prefix="..." %>`
  - `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
  - `<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>`

- **Suggerimento per gli utenti di Tomcat**
  - Copiare *jstl.jar* e *standard.jar* da *examples/WEB-INF/lib*
  - Anch'essi sono sviluppati sotto il progetto Jakarta

JSTL
JSP Standard Tag Library

- JSTL esposta come diverse librerie di tag
  - Per identificare chiaramente l'area funzionale che coprono
  - Per dare a ciascuna area il suo spazio di nomi

Area	Subfunction	Prefix
Core	Variable Support	c
	Flow Control	
	URL Management	
	Miscellaneous	
XML	Core	x
	Flow Control	
	Transformation	
I18n	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

JSTL
Azioni general-purpose

- Il tag *set* imposta il valore di una variabile
  - Se la variabile non esiste, viene creata
  - La variabile con scope può essere impostata dall'attributo *value*:
  - `<c:set var="foo" scope="session" value="..."/>`
  - O dal corpo dell'elemento:
  - `<c:set var="foo">...</c:set>`

- Il tag *set* può anche impostare una proprietà di un oggetto
  - `<c:set value="value" target="${objectName}" property="propertyName"/>`

- Per eliminare una variabile, c'è il tag *remove*
  - `<c:remove var="foo" scope="session"/>`

JSTL
Azioni condizionali

- Il tag *if* permette l'esecuzione condizionale del suo corpo, a seconda del valore dell'attributo *test*
  - `<c:if test="${cart.numberOfItems > 0}">...</c:if>`

- Il tag *choose* permette l'esecuzione condizionale di vari blocci incapsulati nei suoi sotto-tag *when*
  - Esegue il corpo del primo tag *when* la cui condizione di *test* sia verificata
  - Se nessuna delle sue condizioni di *test* viene valutata a true, allora se presente viene eseguito il corpo del tag *otherwise*
  - ```
    <c:choose>
      <c:when test="${user.category=='trial'}">...</c:when>
      <c:when test="${user.category=='member'}" >...</c:when>
      <c:otherwise>...</c:otherwise>
    </c:choose>
    ```
    ```

JSTL
Azioni cicliche

- Tag *forEach* per iterare su una collezione di oggetti
  - Specificare la collezione tramite l'attributo *items*; l'elemento corrente è disponibile in una variabile denominata tramite l'attributo *var*
  - È supportato un gran numero di tipi di collezione
    - Tutte le implementazioni di *java.util.Collection* e *java.util.Map*
    - *Gli array di oggetti come pure gli array di tipi primitivi*
    - Le implementazioni di *java.util.Iterator* e *java.util.Enumeration*
    - Gli oggetti *java.lang.String,* se contengono liste di valori separati da virgola, per esempio: *"Monday,Tuesday,Wednesday,Thursday,Friday"*

```
<table>
  <c:forEach var="product" items="${cart.products}"><tr>
    <td>${product.name}</td>
    <td>${product.quantity}</td>
  </tr></c:forEach>
</table>
```

JSTL
Parametri della richiesta

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <title>Parameters example</title>
</head>
<body>

<i>Hello, ${param.user}!</i><br/>

<c:forEach var="paramName"
   items="${pageContext.request.parameterNames}">
  ${paramName} = ${param[paramName]}<br/>
</c:forEach>

</body>
</html>
```

JSTL
Header della richiesta

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <title>Headers example</title>
</head>
<body>

<c:forEach var="headerName"
    items="${pageContext.request.headerNames}">
  ${headerName} = ${header[headerName]}<br/>
</c:forEach>

</body>
</html>
```

JSTL
Azioni SQL

- Il tag *query* è usato per eseguire query SQL, che restituiscono insiemi di risultati (JDBC result sets)
  - L'interfaccia *Result* è usata per recuperare le informazioni sul risultato della query, in particolare la proprietà *rows*
  - Per query sql parametrizzate, si usa un tag *param* innestato
  - ```
    <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"
    %> …
    <sql:query var="users">
      SELECT * FROM users WHERE country = ?
      <sql:param value="${param.country}" />
    </sql:query>
    <table>
      <c:forEach var="row" items="${users.rows}"><tr>
        <td>${row.lastName}</td>
        <td>${row.firstName}</td>
        <td>${row.address}</td>
      </tr></c:forEach>
    </table>
    ```
    ```

JSTL
Azioni SQL

- Il tag *setDataSource* permette di impostare la sorgente di dati, ossia di collegarsi al database
  - Si possono specificare attributi per il *DriverManager*
  - url, driver, user, password
  - Mysql
  - ```
    <sql:setDataSource driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://127.0.0.1/reti" user="root" />
    ```
- Il tag *update* è usato per aggiornare righe di un database
- Il tag *transaction* è usato per eseguire una serie di istruzioni sql in maniera atomica
    ```

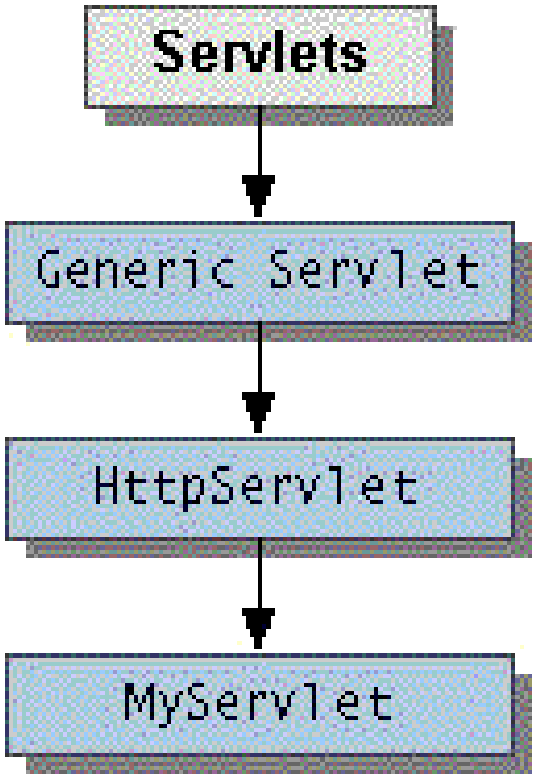
JSTL Funzioni

Area	Function	Tags	Prefix
Functions	Collection length	`length`	

	String manipulation	`toUpperCase, toLowerCase`
		`substring, substringAfter, substringBefore`
		`trim`
		`replace`
		`indexOf, startsWith, endsWith, contains, containsIgnoreCase`
		`split, join`
		`escapeXml`
			`fn`

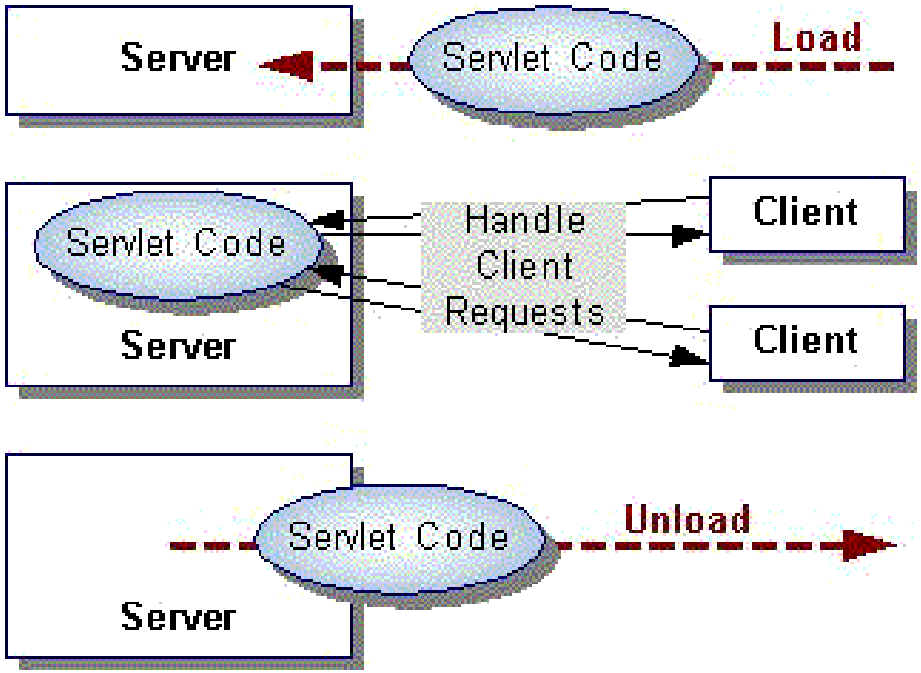
Servlet
Servlet API

- Il package *javax.servlet* fornisce interfacce e classi per scrivere servlet
- L'astrazione centrale nelle API è l'interfaccia *Servlet*
  - Tutte le servlet implementano questa interfaccia, direttamente oppure estendendo una classe che la implementa, come *HttpServlet*
  - *Servlet* dichiara (ma non implementa) metodi per gestire la servlet e le sue comunicazioni con i client
  - Gli autori di servlet devono implementare questi metodi



Servlet
Ciclo di vita

- Inizializzazione
  - Il server carica e inizializza la servlet
- Esecuzione
  - Quando arriva una richiesta
    - Si riceve un oggetto *ServletRequest* con tutte le informazioni sulla richiesta
    - Si usa un oggetto *ServletResponse* per restiruire la risposta
- Distruzione
  - Il server rimuove la servlet
    - Alcuni server fanno questa operazione solo quando vengono chiusi



Servlet Hello user!

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

public class HelloUserServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
   HttpServletResponse response) throws IOException,
   ServletException {
       String user = request.getParameter("user");
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       out.println("<html>");
       out.println("<head><title>Hello Servlet</title></head>");
       out.println("<body><h1>Hello " + user + "!</h1></body>");
       out.println("</html>");
   }
}
```

Servlet
Sottomettere una richiesta GET

```
<form action="http://myhost.com/mycontext/HelloUser"
  method="get">
User: <input type="text" name="user" value="" size="10"
  maxlength="10" /><br />
<input type="submit" name="okbutton" value="OK, submit!" />
</form>
```

Servlet
Aggiungere parametri ai link

- Quando si clicca su un collegamento su una pagina web, il browser invia una richiesta GET relativa alla risorsa
- Si possono aggiungere alla richiesta dei parametri
  - Coppie nome/valore
  - Devono essere aggiunti alla url della richiesta, dopo un carattere "*?*"
  - Se si passano più parametri, devono essere separati da un "*&*"
  - I caratteri *?*, *&* e *BLANK* sono codificati in modo standard

```
<a href="http://myhost.com/mycontext/HelloUser?user=mic&age=34">
  Go to the welcome page
</a>
```

Servlet
Gestire una richiesta POST

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

public class HelloUserServlet extends HttpServlet {
   public void doPost(HttpServletRequest request,
   HttpServletResponse response) throws IOException,
   ServletException {
       String user = request.getParameter("user");
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       out.println("<html>");
       out.println("<head><title>Hello Servlet</title></head>");
       out.println("<body><h1>Hello " + user + "!</h1></body>");
       out.println("</html>");
   }
}
```

Servlet
Sottomettere una richiesta POST

```
<form action="http://myhost.com/mycontext/HelloUser"
  method="post">
User: <input type="text" name="user" value="" size="10"
  maxlength="10" /><br />
<input type="submit" name="okbutton" value="OK, submit!" />
<input type="reset" value="Whoops - erase that" />
</form>
```

JSP Architettura

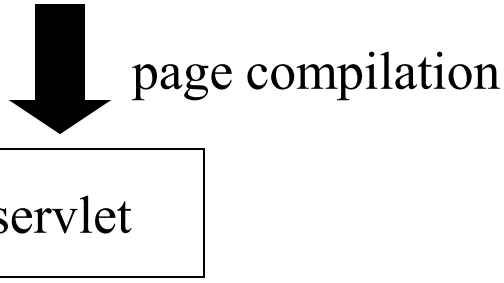
- **Evitare scriptlet e codice java**
  - Non lasciarsi trasportare, non inserire codice java in pagine jsp
  - Incapsulare computazione e logica applicativa in componenti (bean)
  - Preferire espressioni EL e azioni JSTL
  - A differenza delle servlet, le JSP non sono solo per programmatori
  - I disegnatori di pagine web possono essere più coinvolti

- Tipicamente, le pagine JSP sono soggette a una fase di traduzione e una fase di gestione delle richieste
  - La fase di traduzione è compiuta una sola volta, a meno che la pagina JSP non cambi, nel qual caso viene ripetuta
  - Il risultato è un file di classe che implementa l'interfaccia Servlet

JSP
Compilazione di una pagina

```
<%@ page import="java.util.*" %>
<html>
<head>
  <title>Current time</title>
</head>

<body>
   Current time is <%= new Date() %>.
</body>
</html>
```



Servlet container

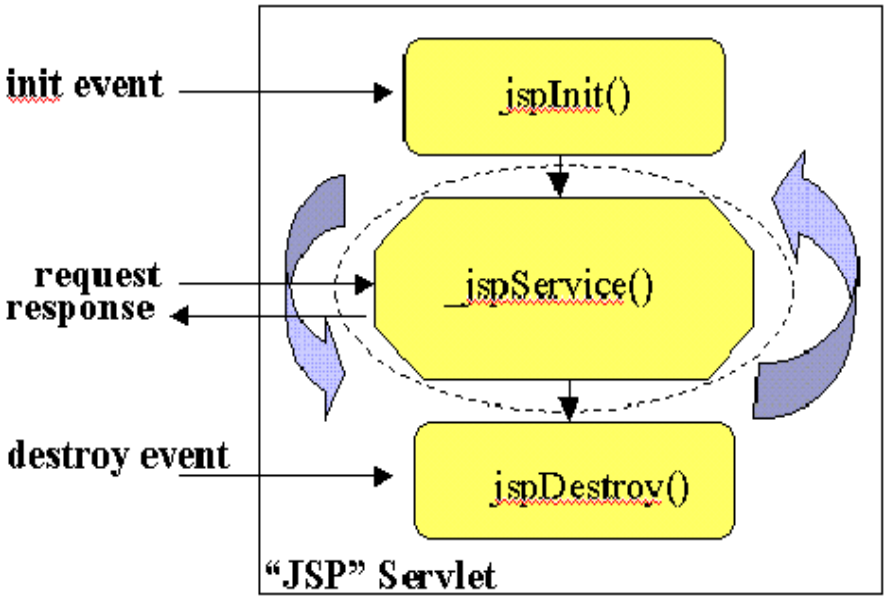
JSP
Servlet generata

```
package jsp;
import [...]
import java.util.*;

public class _0005[...]_ extends HttpJspBase {
  [...]
  public void _jspService(HttpServletRequest request,
      HttpServletResponse response)
      throws IOException, ServletException {
    try {
      response.setContentType("text/html");
      JspWriter out = pageContext.getOut();
      [...]
      out.write("<html>\r\n<head>\r\n  <title>Current
        time</time>\r\n</head>\r\n\r\n<body>  Current time is ");
      out.print(new Date());
      out.write(".\r\n</body>\r\n</html>");
    } catch (Exception e) { [...] }
  }
}
```

JSP
Ciclo di vita

- La classe di implementazione di una pagina jsp estende *HttpJspBase*, che a sua volta implementa l'interfaccia *Servlet*
  - Il metodo *_jspService* sostanzialmente scrive in output il contenuto della pagina jsp
  - Il metodo non può essere sovrascritto, ma lo sviluppatore può gestire gli eventi di inizializzazione e distruzione implementando i metodi *jspInit* e *jspDestroy*
  - Dopo che la classe è caricata, il metodo *_jspService* è responsabile delle risposte ai client



Scriptlet (da evitare)

- I frammenti di codice (*scriptlet)* sono racchiusi tra i tag <% ... %>
- Il codice è eseguito quando la jsp risponde alle richieste
- Si può inserire qualsiasi codice java valido
- Uno scriptlet non è limitato ad una sola riga di codice

- Per esempio il seguente codice (illeggibile!) visualizza la stringa “Hello” tra tag h1, h2, h3, h4:

```
<% for (int i = 1; i <= 4; i++) { %>
  <h<% out.print(i); %>>Hello</h<% out.print(i); %>>
<% } %>
```

Espressioni scriptlet (da evitare)

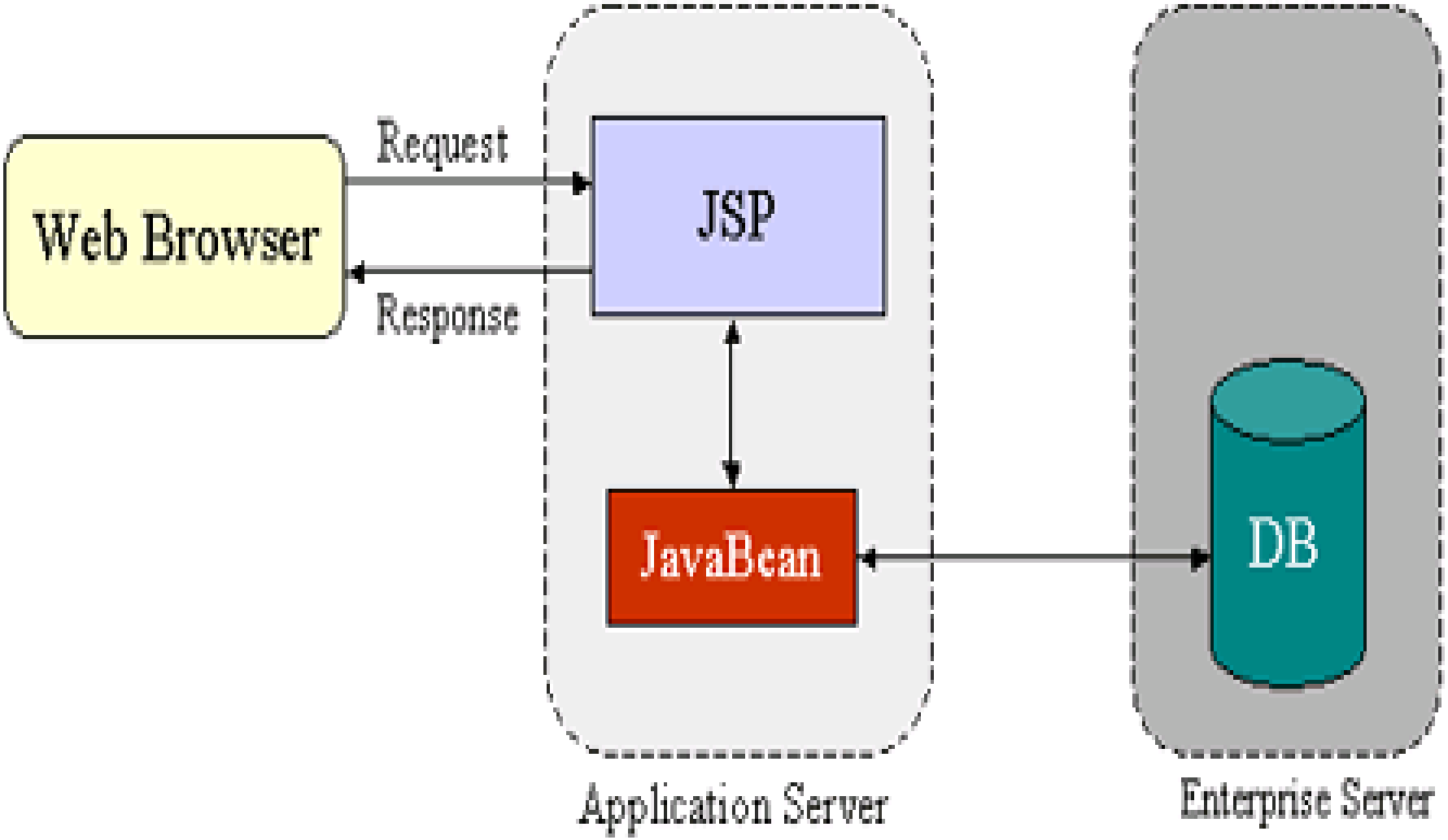
- Le espressioni di sciptlet sono racchiuse tra i tag <%= ... %>, **senza** il punto e virgola finale
  - `<%= new Date() %>`
  - `<%= fooVariable %>`
  - `<%= fooBean.getName() %>`
- Il risultato della valutazione di una espressione è convertito in una stringa e incluso nella pagina (stream) di output
- Le espressioni sono di solito usate per visualizzare semplicemente il valore di una variabile o il valore restituito dall'invocazione di un metodo

```
<% for (int i = 1; i <= 4; i++) { %>
  <h<%= i %>>Hello</h<%= i %>>
<% } %>
```

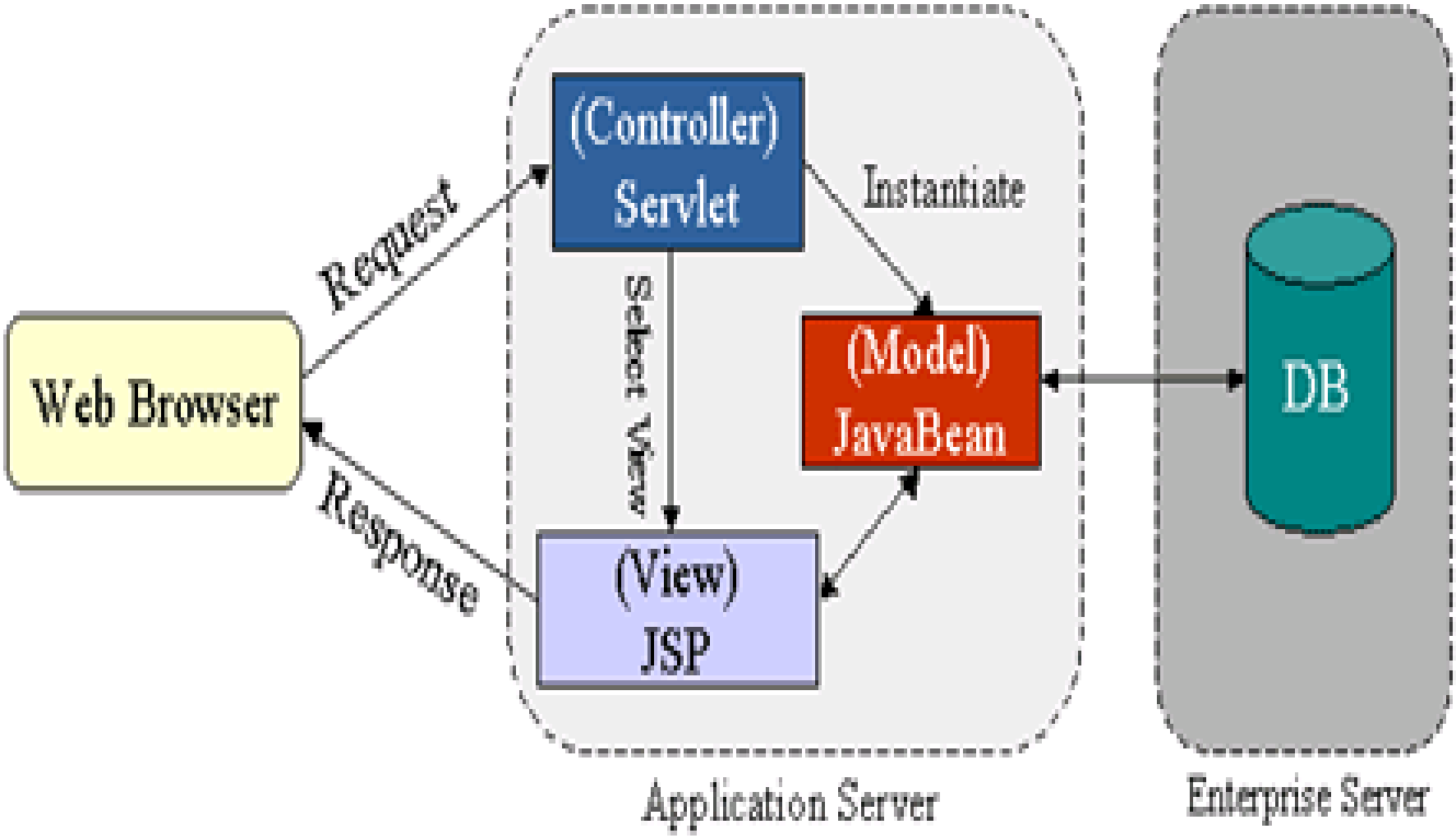
Dichiarazioni (da evitare)

- Le dichiarazioni si trovano tra i tag *<%! ... %>*
- Permettono di definire campi e metodi a livello di pagina
- Il contenuto deve essere codice java valido
- Le dichiarazioni di campi terminano con un punto e virgola
  - `<%! int i = 0; %>`
- Si possono anche dichiarare metodi
  - Per esempio si può sovrascivere il metodo di inizializzazione
  - ```
    <%! public void jspInit() {
      //some initialization code
    }%>
    ```
    ```

Request chaining
Semplice design pattern



Request chaining Design pattern MVC



Request chaining Servlet → JSP

- Elaborazione della richiesta nella servlet intermedia
  - Può accedere a eventuali bean associati alla richiesta
  - Può creare ulteriori bean
  - Può inoltrare la richiesta usando il *RequestDispatcher*

```
public void doPost (HttpServletRequest request,
    HttpServletResponse response) {
  try {
    FormBean f = (FormBean)request.getAttribute("fBean");
    f.setName("Mogambo");
    // do whatever else necessary
    getServletConfig().getServletContext().
      getRequestDispatcher("/jsp/Bean2.jsp").
      forward(request, response);
  } catch (Exception ex) { ... }
}
```

Request chaining Servlet → JSP

- La pagina JSP può accedere alle proprietà dei bean per presentare i risultati

- ```
  <jsp:useBean id="fBean" class="govi.FormBean" scope="request"/>
  <jsp:getProperty name="fBean" property="name" />
  ```

- La richiesta può essere ulteriormente inoltrata ad altre pagine JSP o servlet, con una azione *forward*, o *include*

  - ```
    <jsp:forward page="/servlet/JSP2Servlet">
      <jsp:param name="name1" value="value1" />
    </jsp:forward>
    ```

  - ```
    <jsp:include page="another-page.jsp">
      <jsp:param name="name1" value="value1">
    </jsp:include>
    ```
    ```

Tomcat Introduzione

Implementazione open-source delle tecnologie Servlet e JSP

Sviluppata sotto il progetto Jakarta dalla fondazione Apache

- Integrato da Sun nella implementazione di riferimento J2EE
- Specifiche Servlet e JSP sviluppate secondo il Java Community Process
- Contributi di Apache, Sun e altre aziende

Tomcat
Principali cartelle

- */bin* – Vari script per avvio, chiusura ecc. File **.sh* (per sistemi Linux/Unix) e **.bat* (per sistemi Windows)
- */conf* – File di configurazione. *server.xml* per la configurazione del container
- */lib* – Librerie comuni. *servlet-api.jar* deve essere aggiunta al classpath java per compilare le servlet
- */logs* – File di log
- ***/webapps*** – Qui è dove vanno messe le nostre applicazioni web. Bisogna creare una nuova sottocartella per ogni nuova applicazione web

Tomcat
Organizzazione delle cartelle

- Bisogna organizzare i file delle applicazioni web come previsto dal formato WAR (Web Application Archive)
- Creiamo dentro *webapps* una sotto-cartella (es. “reti”) per la nostra applicazione web. Lavoreremo al suo interno:
  - **.html, *.gif, *.jsp, *,js, *.css, etc.* – Per le applicazioni più semplici, si possono tenere tutti i file visibili al browser nella radice
  - ***/WEB-INF/* – Tale cartella deve sempre esistere (es. dentro “reti”) anche se vuota; notare il nome in maiuscolo**
  - */WEB-INF/web.xml* – Il descrittore dell’applicazione
  - */WEB-INF/classes/* – Contiene le classi richieste dall’applicazione
    - File *.class e risorse associate (sia servlet che altro) non inclusi in file JAR
    - Come sempre, occorre che le cartelle riflettano l’organizzazione delle classi in package
  - */WEB-INF/lib/* - File JAR necessari, ad esempio librerie di terze parti, driver per database ecc.

Tomcat
Semplice descrittore (web.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app ... >
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>HelloUserServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/HelloUser</url-pattern>
    </servlet-mapping>
</web-app>
```