



Alma Mater Studiorum Università di Bologna

Scuola di Ingegneria

Tecnologie Web T
A.A. 2019–2020

Esercitazione 4

Servlet e Java Server Pages

Home Page del corso: <http://www-db.disi.unibo.it/courses/TW/>

Versione elettronica: L.04.Servlet+JSP.pdf

Versione elettronica: L.04.Servlet+JSP-2p.pdf

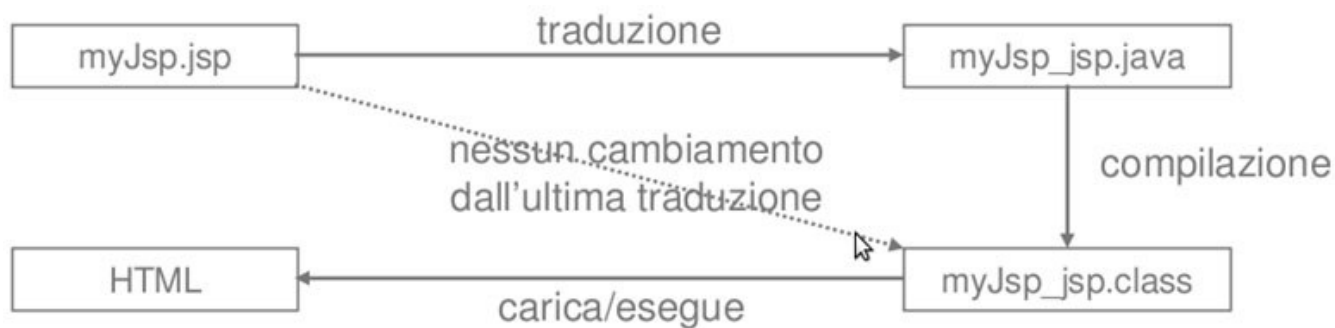
Agenda

- Java Server Pages (JSP)
 - Piccolo riepilogo
- Altro esempio più articolato di esercizio servlet, combinato con pagine JSP già pronte
- Java Server Pages (JSP)
 - Importazione e modifica di un progetto di esempio
 - class-path a tempo di compilazione ed esecuzione
 - deployment ed esecuzione
 - descrittore *web.xml*
 - interazione con l'applicazione
 - servlet e mantenimento dello stato
 - Per approfondire
 - ulteriori esempi

Pagine JSP: piccolo riepilogo

- **Pagine HTML con estensione *.jsp* che includono codice Java**
 - trasformate dal Servlet Container in classi Java che estendono `javax.http.HttpServlet`
 - in Tomcat ciò avviene attraverso una particolare Servlet mappata sulle risorse `*.jsp`, detta `JspServlet`
- Attraverso l'esecuzione di codice Java, il Web server permette di ottenere
 - contenuto HTML generato dinamicamente
 - side-effect quali esecuzione di logica di business complessa, scritture su database, ecc...
- L'insieme dei blocchi di codice Java all'interno di una pagina JSP **contenuti in JSP scriptlet** deve costituire un insieme di istruzioni ben formato
 - possibile apertura e chiusura di parentesi graffe in blocchi distinti di codice Java, separati da codice HTML
 - effetto *simile* a quello ottenibile attraverso un linguaggio di scripting interpretato...
 - ...in realtà istruzioni compilate lato server, prima della loro esecuzione

Ciclo di vita e costrutti principali



- **Direttive** `<%@ %>` o `<jsp:directive.name attribute />`
 - **proprietà generali della pagina**, importazione di nomi di classe, uso della sessione, ecc...
 - processate a tempo di compilazione della JSP in Servlet
- **Espressioni** `<%= ... %>` o `<jsp:expression> java expression </jsp:expression>`
 - trasposizione del risultato della **valutazione di espressioni Java direttamente nel codice HTML** prodotto dalla pagina
 - *n.b.*: permettono la valutazione di espressioni (che restituiscono un risultato), *non* di istruzioni (quindi niente ';' finale)
- **Scriptlet** `<% ... %>` o `<jsp:scriptlet> java instructions </jsp:scriptlet>`
 - codice Java la cui valutazione procede insieme all'elaborazione del contenuto della pagina JSP al fine di produrre l'HTML finale...
 - ...ma la cui compilazione avviene ben prima di questo momento (in caso di fallimento, non è possibile mostrare alcun risultato parziale)
- **Dichiarazioni** `<%! ... %>` o `<jsp:declaration> java definitions </jsp:declaration>`
 - definizione di **variabili e metodi** che potranno poi essere usati all'interno di *scriptlet* ed *espressioni*

Ulteriori costrutti

▪ Azioni `<jsp:nomeAzione attributiAzione ... />`

permettono di effettuare operazioni a tempo di esecuzione della richiesta

- **useBean**: istanzia un oggetto conforme alle convenzioni JavaBean e lo rende disponibile al codice che segue tramite un preciso identificativo e un preciso scope di validità
- **getProperty**: restituisce in forma di oggetto la property indicata
- **setProperty**: imposta il valore della property indicata
- **include**: include a request-time (non a compile-time, come le direttive) il contenuto di un file nel sorgente della JSP valutato dal server
- **forward**: cede la gestione della richiesta a un'altra risorsa
- **plugin**: genera il contenuto necessario per scaricare un plug-in Java

▪ Oggetti 'embedded' o 'built-in': risorse immediatamente utilizzabili nel codice della pagina JSP senza dover creare istanze

- **page**: proprietà e caratteristiche della vista corrente
- **out**: flusso di output su cui riversare l'HTML
- **request**: richiesta HTTP ricevuta, suoi attributi e parametri
- **response**: risposta HTTP da produrre e sue proprietà
- **session**: stato dell'utente mantenuto lato server associato alla richiesta corrente
- ...

Gestione inventario di negozio (04a_TecWeb.zip)

Si parta da un'applicazione Web esistente basata su pagine JSP già realizzate:

- pagina **gestioneCliente.jsp** visualizza merce attualmente selezionata dal cliente che un commesso sta servendo
 - permette di **aggiungere nuovi oggetti** all'insieme della merce selezionata;
 - **vendita al cliente è conclusa quando commesso preme pulsante *concludi***: oggetti selezionati vengono considerati effettivamente venduti e commesso può iniziare a servire un altro cliente. Ciascuna vendita è relativa a uno specifico giorno e a un insieme di prodotti (identificativo prodotto, quantità venduta e prezzo unitario)
- pagina **statistiche.jsp** permette di effettuare **analisi sulle vendite effettuate (già concluse)**, al fine di calcolare **ricavo complessivo** in un intervallo temporale

Ancora servlet, integrate con JSP

La pagina **statistiche.jsp** in realtà si avvale di una Servlet (*completamente da realizzare*) per effettuare il calcolo; Servlet dovrà restituire il risultato alla pagina **statistiche.jsp** che si occuperà della presentazione all'utente

Servlet

- deve ricevere range temporale (giorno iniziale e finale, estremi compresi) e, opzionalmente, codice numerico dell'oggetto di interesse per il calcolo del ricavo
- se tale codice è omesso, calcolo ricavo viene effettuato su tutti gli oggetti in inventario (nota: *quando la servlet imposta tramite `setAttribute` il risultato, assicurarsi che questo sia di tipo **float**, come richiesto dalla pagina **statistiche.jsp** predisposta*)

Si noti che:

- ciascun commesso deve poter effettuare le proprie ricerche statistiche, considerando le vendite effettivamente concluse da parte di tutti i commessi
- risultato ultima ricerca effettuata da un commesso (insieme ai criteri di ricerca) deve essere visualizzato ogniqualvolta quel commesso ritorna alla pagina **statistiche.jsp** (quindi senza supporto della Servlet)

Primo vero e proprio esercizio su JSP...

- Il file **04b_TecWeb.zip** contiene lo scheletro di un progetto Eclipse **Ant-based** basato sull'uso di pagine JSP
 - contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente
- Importare il progetto come visto nelle precedenti esercitazioni:
 - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*
- Attraverso i target Ant, compilare, creare «pacchetto» e pubblicare sul server l'applicazione Web 'AS IS' e avviare Tomcat
 - ricordarsi di modificare opportunamente il file *environment.properties*

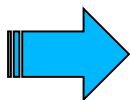
Applicazione Web 04b_TecWeb

- Accedendo all'applicazione Web tramite...

http://localhost:8080/04b_TecWeb

... Servlet Container seleziona automaticamente la risorsa corrispondente alla pagina *index.jsp* per servire la richiesta

- A differenza della scorsa esercitazione, tuttavia...
 - ... la pagina JSP presenta un messaggio di attesa...
 - ... ma **comanda una redirectione** non al browser dell'utente, ma **al proprio Servlet Container**, senza restituire alcuna risposta all'utente
 - Ovviamente questo non è l'effetto desiderato



CORREGGERE!

Il gioco dei forward

■ Seguite il gioco dei forward

- data la complessità che un'applicazione Web può assumere, capita spesso di **suddividere la logica necessaria a servire una richiesta su più componenti**
 - filtri per aprire e chiudere transazioni, servlet per accedere al database, pagine JSP per produrre la vista di risposta, ...
- si migliora la manutenibilità e si evita di replicare parti di codice comuni

■ Qui però occorre sistemare...

- **redirect iniziale**, affinché avvenga tramite un ordine dato al browser, non al server: modificare *index.jsp* tramite l'uso di `<meta http-equiv="..." ... >`, non solo refresh ma anche ridirezione locale al browser
- **mapping** della classe Servlet a cui viene inoltrata la gestione della richiesta: correggere piccolo errore in *web.xml*
- passaggio del **parametro di inizializzazione** richiesto da tale servlet: completare opportunamente *web.xml*

■ Dopodiché occorre ri-eseguire il deploy (Ant)

- le modifiche ai descrittori XML (*web.xml* nel nostro caso) e al codice Java necessitano che il progetto venga ripacchettizzato e ripubblicato sul server
- attendere che **Tomcat riconosca la modifica** e comandi la ripartenza dell'applicazione

Negozio online: *home.jsp*

- Se siete arrivati fin qui, esplorate la struttura della pagina *home.jsp*
 - nell'IDE, per capire come è stato generato HTML finale
 - con Firebug, per capire come è strutturato HTML finale
- Aspetti interessanti da osservare
 - parti comuni a tutte le altre pagine incluse mediante frammenti JSP esterni
 - ogni pagina è in grado di modificare il colore di sfondo della “tab” ad essa corrispondente mediante l'analisi dell'URL richiesto (vedi *menu.jsp*)
 - i soliti “giochi” con i CSS
 - ...
- Benvenuti nel negozio online!
 - **home.jsp** come pagina di benvenuto
 - **catalogue.jsp** per gestire il catalogo della merce in vendita
 - **cart.jsp** per gestire il carrello di un cliente
 - **checkout.jsp** per terminare l'acquisto

Gestione del catalogo: *catalogue.jsp*

- Pagina per gestire il catalogo degli articoli in vendita
 - **molto più complessa e completa** della pagina di benvenuto
 - realizzata per mezzo di un **bean con scope di applicazione**: gli articoli e le corrispondenti quantità disponibili sono concetti “unici”, uguali per tutti gli utenti del negozio
- Direttive
 - errori, sessione, bean utilizzati, import di classi Java
- Dichiarazioni
 - metodi richiamati nel seguito, per aggiungere/rimuovere oggetti
- HTML, scriptlet ed espressioni
 - analisi dei parametri della richiesta per decidere cosa fare
 - layout a due colonne (tramite attributo float)
 - inserimento di nuovi articoli
 - visualizzazione del contenuto attuale del catalogo
 - ogni richiesta per *catalogue.jsp* causa una nuova visualizzazione della pagina
- **Prendetevi un po' di tempo per analizzare il funzionamento (domande?)**

Gestione del carrello: *cart.jsp*

- Pagina per gestire il carrello degli articoli scelti dall'utente/cliente
 - tale selezione è diversa da cliente a cliente: servirà quindi un ***bean con scope di sessione***
- Ricalcando la struttura di *catalogue.jsp*, riuscite a realizzarla voi?
 - sulla sinistra iterate sugli articoli nel catalogo
 - per ogni riga introducete un comando per inviare una richiesta di aggiunta al carrello
 - nella pagina analizzate i parametri della richiesta per capire come gestirla
 - incapsulate i metodi di utilità dentro le dichiarazioni
 - sulla destra mostrate il contenuto corrente del carrello (ogni nuova richiesta determina l'aggiornamento della pagina)

Gestione del checkout: *checkout.jsp*

- Pagina per concludere l'ordine
 - decrementare le quantità nel catalogo
 - salvare la selezione dell'utente

dove?

- nel database!

Come imparerete a fare tra qualche lezione...

Per approfondire...

Tomcat fornisce out-of-the-box alcuni esempi relativi all'utilizzo delle JSP (e anche Servlet), **molto utili** come riferimento

- accessibili a partire da
<http://localhost:8080/examples>
- funzionamento ed estratti del codice sorgente
- il codice sorgente completo è comunque disponibile su file system, nella directory di deployment che corrisponde al contesto “*examples*”

