APPENDICE A

Servlet e *J*ava *S*erver *P*age

A.1 Cosa è una Servlet e come funziona

Una servlet è un particolare tipo di applicazione Java, in grado di essere eseguita all'interno di un web server e di estenderne le potenzialità. Per interagire con una servlet, un client (tipicamente un browser) deve far uso del protocollo http.

Tipicamente il meccanismo di funzionamento di una servlet è il seguente:

1. il client (il browser) richiede ad un web server remoto una pagina html al cui interno è presente un riferimento ad una servlet;
2. il web server invia la pagina richiesta, e manda in esecuzione la servlet, la quale effettua tutta una serie di operazioni ed eventualmente invia al client un risultato che, incapsulato nella pagina html, verrà visualizzato all'interno della finestra del browser;

la servlet quindi viene eseguita da una JVM inserita direttamente nel web server.

A.2 Perché utilizzare una Servlet piuttosto che un CGI ?

La caratteristica più importante di una servlet è che, contrariamente ad un programma standard CGI scritto in PERL o in C, non viene creato un processo ogni volta che il client effettua la richiesta.

Ricordiamo che un Web server quando riceve una richiesta CGI, deve eseguire un programma completamente diverso e restituire il risultato del processo al Web

browser.

A difesa dei CGI possiamo dire che essi sono stati concepiti poco dopo la nascita del Web, in un periodo in cui i Web server ricevevano un numero d'accessi limitato e quindi la velocità di risposta ad ogni richiesta dei client, non era una condizione così rilevante. Con l'aumento degli accessi ad Internet, i Web server devono soddisfare un numero sempre crescente di richieste e spesso devono servire svariati accessi contemporaneamente, facendo sì, che la velocità di risposta, diventi una condizione non trascurabile.

Le Servlet, appunto, ci permettono di ottenere delle performance di esecuzione notevolmente migliori rispetto ai CGI, poiché, l'attivazione di una Servlet non richiede la creazione di un nuovo processo per ogni richiesta, ma è eseguita in parallelo all'interno dello stesso processo del Web server.

Approfondiamo questa caratteristica importante delle Servlet, che le contraddistinguono dai CGI e cioè quella di non richiedere l'attivazione di un nuovo processo.

Una Servlet, facendo parte integrante dello stesso processo del Web server, si può immaginare che sia una procedura dello stesso server.

Questo perché il Web server, effettua un caricamento dinamico della classe Servlet, durante lo start-up, oppure come conseguenza della prima richiesta alla Servlet da parte di un client, dopodiché questi rimane residente in memoria ed a disposizione delle successive richieste. Poiché il tempo di attivazione di una Servlet, in pratica è quello del caricamento dinamico di una qualunque classe e poiché una volta effettuato, non deve essere più ripetuto, si può capire perché, il tempo di "*chiamata*" di una Servlet, è assimilabile a quello di una chiamata ad una procedura all'interno di un processo.

Per comprendere meglio la differenza con i CGI, si consideri che il tempo di attivazione ed inizializzazione di un CGI deve essere calcolato per ogni richiesta del client, mentre il tempo necessario al caricamento ed all'inizializzazione di una Servlet, di per sé minimo, essendo effettuato una sola volta, è ripartito per tutte le successive richieste dei client.

A.3 Metodi principali delle servlet

I due metodi principali di una servlet sono:

- *init()*: rappresenta il momento della creazione ed istanziazione della servlet: come nelle applet, serve per inizializzare tutti i parametri e le variabili da utilizzare per il funzionamento. Nella *init* spesso si ricavano i parametri passati alla servlet dal sistema;
- *service()*: rappresenta la richiesta da parte del client http sotto forma di una GET o POST http (dal punto di vista di una servlet non c'è differenza). I due parametri fondamentali del metodo service sono HttpServletRequest, HttpServletResponse. Il primo rappresenta la richiesta http (con il quale ottenere informazioni sulla richiesta, come i parametri), mentre il secondo identifica la risposta con la quale restituire informazioni al client.

Appare evidente che essendo il metodo *init()* delle Servlet chiamato solamente in fase di inizializzazione, tutte quelle operazioni che non necessitano di esecuzione ad ogni richiesta dei client, si possono inserire proprio in questo metodo. Quindi, l'incremento di prestazioni, confrontato con i CGI, non si limita ai soli tempi di attivazione. La persistenza delle Servlet dopo la sua inizializzazione ci permette, infatti, di gestire molte problematiche in modo più efficiente.

Se si pensa, infatti, alla necessità da parte dei clients di fare accesso ai database, si può subito intuire che l'operazione di connessione al database, che comporta un rilevante dispendio di tempo e che con i CGI è effettuata ad ogni richiesta dei clients, con le Servlet è sufficiente inserirla nel metodo init(), facendo sì che non venga più ripetuta per tutte le successive richieste.

E' evidente, che questa gestione, è più "pulita" e ci permette di ottenere risposte alle richieste dei clients molto veloci.

La deallocazione di una Servlet avviene in genere allo shutdown del Web server, in conseguenza del quale è chiamato il metodo destroy().

Inoltre la deallocazione della Servlet, avviene anche quando si elimina il suo Alias dalla tabella di configurazione dei "Servlet Alias" ed infine, quando si preme esplicitamente il button "Unload" dal menù di gestione Servlet del Web server, funzionalità che ci permette di modificare una Servlet e attivarla senza fare lo shutdown del Web server.

A.4 Gestione di diverse richieste allo stesso servlet

Ogni richiesta ad una Servlet è gestita da un "handler Threads", quindi ci si può trovare nella situazione in cui più "handler Threads" fanno uso concorrente della stessa Servlet. In pratica, ogni richiesta del client attiva un Servlet Threads e molti metodi service() possono essere contemporaneamente attivi.

L'interfaccia Servlet, non assicura una protezione dagli accessi concorrenti e quindi è opportuno, implementare il metodo service(), in modo da proteggere i dati da accessi concorrenti, utilizzando la parola chiave synchronized.

La soluzione più semplice è definire il metodo service come synchronized, ma nel nostro caso, non ne abbiamo la necessità.

Infatti, dovendo effettuare una connessione al database, potremmo trovarci in una situazione di concorrenza se dovessimo definire l'oggetto Connection, come variabile di istanza della classe ed effettuare la connessione al database, all'interno del metodo service(). Poiché la connessione al database è effettuata nel metodo init() ed essendo quest'ultimo chiamato una sola volta, non esistono problemi di concorrenza.

A.5 Cosa è una Java Server Pages ?

Quando si parla di creazione dinamica di pagine HTML, significa generalmente che tutto il lavoro di produzione viene svolto sul lato server, e quindi non sono richieste particolari funzionalità aggiuntive al browser, il quale deve semplicemente essere in grado di interpretare HTML standard.

Le servlet offrono una serie di vantaggi rispetto alle soluzioni concorrenti, legati al fatto che si utilizza Java come linguaggio di sviluppo.

Le servlet però richiedono un po' di lavoro in più per risolvere aspetti legati al life-cycle, al mantenimento dello stato, e passaggio di parametri fra oggetti differenti. Le Java Server Pages sono state proprio pensate con lo scopo di eliminare queste limitazioni, oltre che ad offrire ulteriori importanti funzionalità. Mentre le servlet sono classi Java con immersi tag HTML\XML, le JSP sono pagine HTML/XML all'interno delle quali è possibile inserire codice Java: questo codice verrà poi gestito dal server che provvederà alla sua compilazione ed esecuzione. Questo fatto permette al programmatore di inserire parti di codice Java in modo molto simile a come farebbe con script JavaScript, VBScript o simili. Come conseguenza si ha un ulteriore impulso alle performance ed alla portabilità dato che il codice prodotto viene eseguito sul server, e non nel browser.

Per eseguire una JSP si deve utilizzare un web server in grado di supportare tale tecnologia, oppure far uso di un cosiddetto application server che estenda le funzionalità del web server.

Il meccanismo dell'application server permette di estendere le funzionalità del web server: ad esempio quando un client richiede una risorsa non gestibile dal server direttamente (come JSP o servlet), per mezzo di chiamate di sistema, tale richiesta viene passata all'application server. Questo meccanismo permette flessibilità e modularità.

Le JSP sono la risposta a soluzioni simili come ASP (di cui parleremo nella successiva appendice), ma in chiave Java: oltre ai soliti innegabili vantaggi derivanti dall'utilizzo di Java come linguaggio, in questo caso la portabilità assume un ruolo particolarmente importante.

Il potersi svincolare dalla particolare piattaforma (sistema operativo ed hardware), è una caratteristica molto importante, in un settore (quello dei server web), in continua evoluzione. Inoltre la sempre maggiore stratificazione in atto rende Java ancora più prezioso, dato l'utilizzo di protocolli standard di comunicazione. La portabilità deve essere intesa anche dal punto di vista del client: utilizzare Java come linguaggio di script, che in ogni caso però viene interpretato dal server, potrà forse contribuire alla

diminuzione del proliferare dei molti linguaggi, DHTML, JavaScript e JScript, VBScript, i quali oltre a creare confusione, spesso non sono supportati dai differenti browser in commercio. L'enfasi maggiore che è stata data all'utilizzo dei componenti, facilita il riuso, la portabilità e semplifica lo sviluppo. Questa visione si riallaccia a quella che è l'attuale tendenza della programmazione ad oggetti utilizzata anche in ambiti più ampi.

A.6 Invocazione di una JSP

Esistono due possibilità per accedere ad una Java Server Page:

1. invocazione centralizzata rispetto la JSP descritta nel Capitolo 3;
2. centralizzato rispetto la Servlet: Il client effettua una richiesta verso una servlet, il quale esegue le operazioni del caso, ed incapsula i risultati in un component. Successivamente la servlet provvede a chiamare la pagina JSP che preleva i dati immagazzinati nel component e produce l'output voluto.

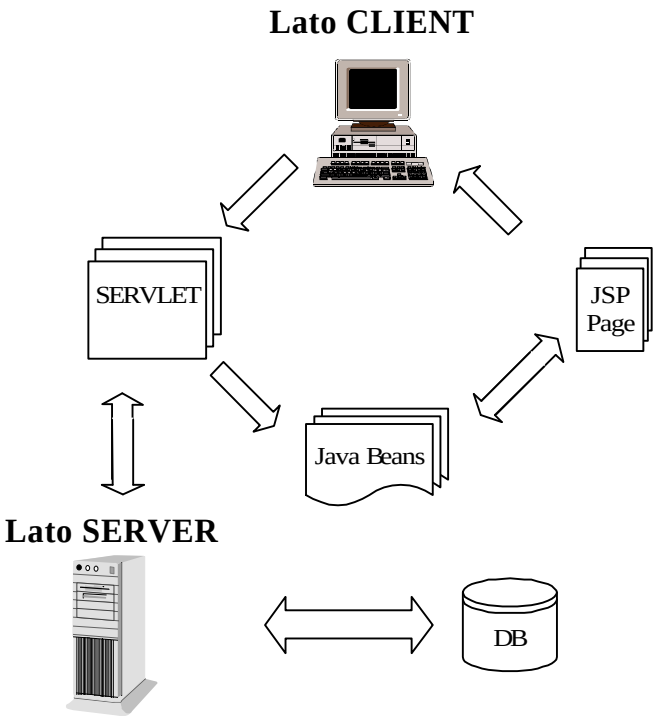

Figura A1:Secondo modello di invocazione di una pagina JSP

A.7 Sintassi delle pagine JSP

L'utilizzo di JSP è molto semplice, dato che si basa fondamentalmente su Java, e HTML: la struttura e la logica infatti si rifanno a quella dell'HTML, mentre la possibilità di inserire codice Java, permette a tutti i programmatori Java una migrazione indolore, senza cioè la necessità di dover imparare un nuovo linguaggio di script. I tag che si possono utilizzare possono essere suddivisi in due categorie:

1. orientati allo script (*Scripting-centric tag*);
2. orientati al componente (*Component-centric tag*): elementi molto simili all'HTML, con i quali è possibile ricoprire la maggior parte delle necessità.

La prima tipologia di tag offerta da JSP, è quella orientata allo scripting, per mezzo della quale è possibile inserire vero e proprio codice script all'interno della pagina.Questi tag si suddividono a loro volta in:

- ***Direttive***: sono particolari istruzioni che avvertono il JSP-Engine (ovvero il web server o l'application server JSP enabled) di una particolare impostazione da utilizzare. La sintassi è <%@ ...%>.
- ***Dichiarazioni***: permettono la definizione di variabili con scope globale rispetto alla pagina. Le dichiarazioni si includono all'interno della coppia di tag <SCRIPT...> </SCRIPT> .
- ***Sciptlets***: vere e proprie porzioni di codice da eseguire. Uno scriptlet deve essere inserito all'interno della coppia di tag <% ... %>.
- ***Espressioni***: denotate dalla sintassi <%= ... %> permettono l'inserimento di vere e proprie espressioni, variabili, valutate run time.
- ***Variabili implicite***: variabili che sono disponibili automaticamente senza la necessità di particolari inclusioni. Sono utilizzabili per mezzo della sintassi <%…%>.

La seconda tipologia di tag offerta da JSP, è quella orientata ai componenti, questi non implicano nessun tipo di script o codice Java. Essendo tag elementari, sono molto simili all'HTML, mentre la loro organizzazione gerarchica si riconduce alla struttura "oggetto.proprietà". Come nel caso dell'HTML, questi tag non sono case-sensitive, e non è obbligatorio utilizzare i doppi apici per la definizione dei valori.Attraverso le JSP è possibile accedere e manipolare componenti (JavaBeans) all'interno della pagina stessa: questo si traduce poi nella possibilità di accedere ai metodi e proprietà degli stessi, e tipicamente questo meccanismo avviene per mezzo di una scrittura del tipo:

```
nomeBean:nome_proprietà
```

come ad esempio

```
myUserBean:address
```

Si noti che l'accesso ai campi (proprietà) del bean avviene per mezzo del carattere ":" e non ".": questo allo scopo di evidenziare il fatto che non si tratta di codice Java, ma di script JSP. Oltretutto se fosse codice Java, oltre che impossibile, sarebbe formalmente scorretto, accedere alle proprietà del componente direttamente senza l'ausilio di metodi get/set.Vediamo adesso quali sono i tag più importanti per l'utilizzo di un componente all'interno della pagina. Per prima cosa i tag di specificazione del bean:

la coppia <USEBEAN> e </USEBEAN>, per mezzo della quale è possibile rendere disponibile il componente a tutte le invocazioni successive all'interno della pagina. Deve essere utilizzata ad inizio pagina.

<SETONCREATE ...>, è una specie di *init()* e serve per settare alcune proprietà al momento della creazione del componente; può essere utilizzato solo all'interno della coppia <USEBEAN> ,</USEBEAN>.

<SETFROMREQUEST ...> serve per passare al bean un parametro arrivato come richiesta dall'esterno (con il meccanismo tipico di una get CGI); può essere utilizzato solo all'interno della coppia <USEBEAN> ,</USEBEAN>.

Per la visualizzazione delle proprietà di un bean invece possiamo utilizzare:

<DISPLAY ... > che restituisce il valore di una proprietà a valore singolo
<LOOP... > </LOOP> che fornisce un meccanismo ripetitivo per la visualizzazione di proprietà multiple.

A.8 Metodi definibili nelle pagine JSP

Una pagina Jsp è eseguita attraverso un JSP container, il quale è istallato sul Web Server. Il JSP container trasferisce la richiesta del client alla pagina JSP e risponde attraverso la stessa JSP.

All'interno della pagina JSP possono essere definiti metodi che possono essere invocati da qualsiasi oggetto definito all'interno della pagina. Il JSP Container quando un client invoca una JSP non fa altro che generare una servlet al cui interno è sempre presente il metodo _jspservice.

All'autore della JSP, oltre a poter definire dei nuovi metodi è anche permesso di indicare alcune azioni che devono essere eseguite quando vengono invocati i metodi init() e destroy() della servlet generata. Questo si traduce nel poter ridefinire due metodi jspinit e jspdestroy.

La prima cosa che viene fatta quando viene richiesta una pagina JSP è quella di invocare, se presente, il metodo jspinit() che ha il compito di inizializzare la pagina. In modo analogo il JSP container può in caso di necessità, attraverso la pagina JSP, richiamare risorse che non sono utilizzate per un lungo periodo invocando, se presente, il metodo jspdestroy().

Nella figura A2 vengono messe in evidenza le relazioni che intercorrono tra la pagina JSP e il JSP Container.

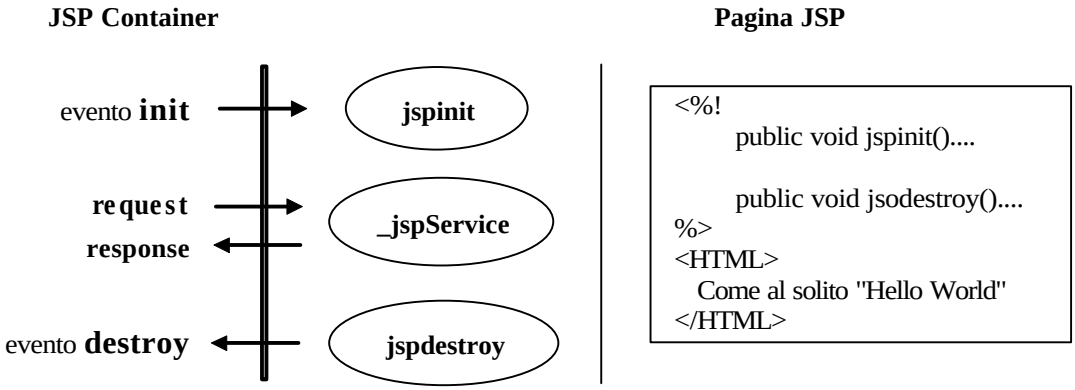

Figura A2: Relazioni tra la pagina JSP e il JSP container

Nella figura seguente vengono illustrati i metodi attraverso i quali il JSP Container processa le pagine JSP:

Metodi invocati dal JSP Container	Commenti
void **jspinit**()	Metodo opzionale che può essere o meno definito all'interno della pagina JSP. Viene invocato quando la pagina JSP è inizializzata. Quando è invocato tutti i metodi nella servlet, incluso getServletConfig() sono disponibile.
void **jspdestroy**()	Metodo opzionale che può essere o meno definito all'interno della pagina JSP. Viene invocato prima di distruggere la pagina.
void **_jspService**(..) throws IOException	Metodo non ridefinibile all'interno della pagina JSP. Viene generato automaticamente dal JSP Container, basandosi sul sul contenuto della pagina JSP. il metodo viene invocato ad ogni richiesta del client.

Figura A3: Metodi attraverso i quali il JSP Container processa le pagine JSP