

Java Server Side JSP

Ing. Cesare Monti - 27 aprile 2005

cosa vedremo

- JSP
 - storia
 - contesto
- JSTL

JSP: il contesto

- siamo sul web, hic sunt:
 - server web
 - DB
 - beans, bundles,
 - ... oggetti in generale ...
 - agenti ... tra non molto

JSP: il contesto

- all'epoca la prima sfida fu quella di aggiungere computazione lato client
- Java fece le Applet ... con i relativi problemi di:
 - esecuzione
 - affidabilità
 - sicurezza
 - Microsoft Java Machine (quella di allora!)
- all'epoca tempi lunghi per spostare codice
 - non c'era la banda larga
- poche applicazioni grafiche per contenere le dimensioni

JSP: il contesto

- si cercavano quindi "maniere per uniformare"
 - e si tornò al server side
 - prima
 - con maniere standard di far scrivere output (html) da classi java (servlet)
 - poi
 - con nuove maniere di scrivere "web",
 - java dentro html

JSP: il contesto

- a bit of history: Model Vs Model
 - con le servlet la sfida fu di abilitare "automi software" per la scrittura di html
 - con jsp la sfida cambiò focus, si cercava di scrivere codice html (tag) a cui fosse associato codice java da eseguire.
 - di quale sia l'impatto di questo cambio di focus ne discuteremo più avanti

JSP: il contesto

- per riuscire nel tutto occorre qualche anno
- si realizzarono i "container" conformi alle API JSP
 - Apache Jakarta Tomcat
 - Resin servlet / JSP container
 - WebSphere Application Server
 - GNUJSP
 - Jetty

JSP: il contesto

- ... e le stesse specifiche JSP cambiarono:
 - dalla API 1.0 ...alle 2.0
 - ... ora le 2.4
 - con tanti pro e qualche contro

JSP: il contesto

- volendo fissare il contesto:
 - JSP opera server-side
 - associa codice java a tag HTML
 - interagisce con java in ogni sua forma
 - ha bisogno di un ambiente consono alla sua vita

JSP: il contesto

- le pagine JSP abbisognano di un contenitore adatto
 - perchè ?
 - ... se c'è del codice qualcuno lo deve compilare e visto che è java pure interpretarlo
 - chi lo fa?
 - (retorica) il container o engine che dir si voglia

JSP: il contesto

- guardiamo il container:
 - deve stare sul web,
 - rispondere a richieste http (sicuramente un server web)
 - compilare e interpretare java (sicuramente avrà un JVM)
 - un esempio per tutti:
 - Jakarta Tomcat

JSP: il contesto

- Tomcat
 - nasce come progetto Open Source
 - nasce come una falange di apache ... poi diviene un server web a se stante
 - <http://jakarta.apache.org>
 - ... è alla sua quinta release (2004)
 - è più lento di apache nella risposta http normale
 - ... ma risponde anche all'elaborazione di jsp e pare sia meglio degli altri

JSP: il contesto

- Tomcat
 - prevede una struttura pubblica delle pagine organizzate in directory
 - come apache
 - compila le pagine alla prima loro invocazione
 - le interpreta ogni volta che vengono chiamate

JSP: la sintassi

- si è detto java dentro html ergo ci sono dei tag di marcatura
- `<%` (apro il tag)
- `%>` (chiudo il tag)
 - `<!-- questo è un esempio di un commento -->`

JSP: la sintassi

```
<html>
<head><title>Prima pagina JSP </title></head>
<body><table><tr>
  <% for (int i= 0; i<=3; i++){
    %> <td>
      <%= "&nbsp;" + java.util.date() + "&nbsp;";%>
    </td> <%
  } %>
</tr></table></body></html>
```

JSP: la sintassi

- l'esempio di prima fa pensare...
 - uso un package ... di solito in java li importo ... cosa devo fare qui?
 - lo stesso
- esiste una direttiva di import dei packages
 - che nell'esempio di prima non c'era!
 - `<%@ page import="java.util.*" %>`
 - ha validità solo per la pagina così come in java ha validità per la classe in cui la si usa
 - dicansi ... direttive di pagina

JSP: la sintassi

- Le direttive di pagina sono ciò che ci permettono di specificare il dominio e l'ontologia della pagina jsp
- `<%@ page language="italian"%>`
- `buffer="dimensione del buffer dell'output"`
- `pageEncoding = "codifica dei caratteri della pagina"`
 - `<%@page pageEncoding="UTF-8"%>`
- `errorPage="URL pagina di risposta in errore"`
- `contentType="tipo MIME (Multipurpose Internet Mail Extension)" %>`
 - es: `<%@page contentType="text/html"%>`

JSP: la sintassi

- oltre alle direttive di pagina esistono altri due marker molto comodi
 - `<%@ taglib`
 - per l'inclusione di librerie di tag
 - ... di cui parliamo dopo
 - `<%@ include`
 - per l'inclusione di "pagine" esterne a quella di definizione
 - ... tra un po' vediamo esempi

JSP: la sintassi

- ... e le dichiarazioni di carattere globale?
- `<%! String stringa="Hello World" %>`
- questa viene vista in tutto ciò che viene a "contatto" con la pagina in cui è inserita
 - sia le possibili classi importate a priori
 - sia quello incluse a posteriori

JSP: la sintassi

- la slide precedente è valida anche per le funzioni

```
<%! int somma(int a, int b){
    return a+b;
} %>
```
- e anche in questo caso la funzione è vista da tutto ciò che viene a contatto con la pagina

JSP: la sintassi

- ... per la modularità possiamo includere file esterni
- a compile-time
 - `<%@ include file="URL"%>`
 - l'URL può essere assoluto o relativo alla pagina che lo invoca
- a request-time
 - `<jsp:include page="URL" flush="true"/>`
 - ...che non è più una direttiva di pagina ma un tag a tutti gli effetti,
 - posso specificare anche i parametri da passare alla pagina (per coerenza con il protocollo CGI)
 - `<jsp:include page="URL" flush="true">`
 - `<jsp:param name="first" value="Hello" />`
 - `<jsp:param name="second" value="World" />`
 - `</jsp:include>`

JSP: la sintassi

- la direttiva `<jsp:include page="">`
 - è in realtà un tag personalizzato rilasciato dalla SUN
 - alla sua invocazione viene invocato del codice che esegue una determinata funzione
 - ... l'inclusione appunto
 - a differenza della direttiva di pagina `<%@page include file="">`, il tag include l'output della pagina specificata e non il sorgente

JSP: la sintassi

- in realtà di tag che iniziano con `<jsp:`
 - che hanno quel namespace
- ne esiste una serie:
 - `<jsp:attribute`
 - `<jsp:body`
 - `<jsp:element`
 - `<jsp:fallback`
 - `<jsp:forward`
 - `<jsp:getProperty`
 - `<jsp:plugin`
 - `<jsp:setProperty`
 - `<jsp:useBean`
- ad ognuno di questi è associato codice java per realizzare determinate funzioni

JSP: la sintassi

```
...
<form name="form1" method="GET | POST" action="Azione.jsp">
<input type="text" name="user">
<input type="password" name="pw">
<input type="submit" name="invia dati">
</form>
...
```

JSP: la sintassi

```
<%@ taglib uri="WEB-INF/tlds/html.tld" prefix="html"%>
```

...

```
<td> <html:requestParameter property="user"></td>
```

```
<td> <html:requestParameter property="pw"></td>
```

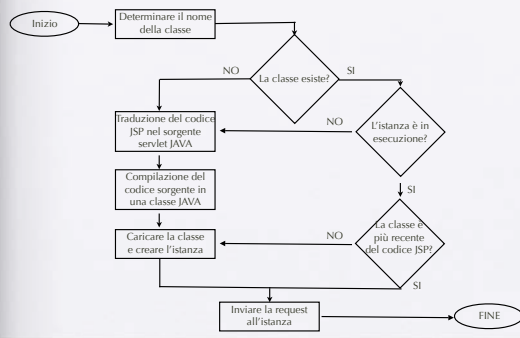
...

- e qui ho già recuperato i valori passati dalle form ... operazione che in CGI avrebbe richiesto molta più fatica
- ho inoltre scoperto che esistono tag o librerie di tag per manipolare l'output

JSP e JAVA

- come già detto, JSP presenta tag che eseguono codice,
 - come è possibile ?

JSP e JAVA



JSP e JAVA

- quindi? io scrivo jsp o java?
 - in realtà io scrivo jsp, che viene tradotto in java dal container in cui lo piazco
 - Tomcat non fa altro che creare servlet senza che l'utente se ne accorga

JSP e JAVA

- ... quindi scrivo servlet?
 - si ... ma in maniera trasparente
 - ... me ne astraggo!
 - no! scrivo html con java dentro ... che viene trasformato in servlet
 - entrambe le risposte sono accettabili
 - posso scrivere servlet e farle interagire con classi e jsp
 - così riutilizzo codice
 - ... ma prima devo capire bene cosa fa jsp e cosa fa una servlet

JSP e BEANS

- "Any Java class that adheres to certain property and event interface conventions can be a Bean." (SUN - 1997 - JavaBeans Technology Survey)
- Dato il nome i beans
 - (la tecnologia si chiama Enterprise Java Beans)
- fanno parte di J2EE
 - e sono quindi applicazioni server side

JSP e BEANS

- posso utilizzarli
- sempre sulla macchina server
 - per sincronia
 - a mo' di contenitore di flag di segnalazione
 - per disciplinare attività
 - se associamo un bean ad un utente quando questi fa login ...
 - per comunicazione
 - lo creiamo e lo poniamo a livello application
 - ...

JSP e BEANS

- sull'uso dei BEANS in rete trovate chilogrammi di letteratura che
 - vi guida
 - vi abilita
 - vi disciplina all'uso
- sono semplici e incredibilmente potenti

JSP e BEANS

- o distribuirli tramite container
- a livello di modello non sono diversi dagli altri
 - basta pensare alle funzionalità che hanno i beans e paragonarle nel contesto web
- la differenza la fa il tipo di bean che utilizziamo
 - session
 - ogni session bean contiene i dati relativi alla sessione corrente sul dato server
 - entity
 - il bean più semplice che può essere serializzato
 - message
 - i web services ne usano parecchi

JSP e LEGACY

- abbiamo già detto che si può riutilizzare qualsiasi classe dentro ad una pagina jsp
- basta dichiararla globale
- `<%! %>`

JSP e LEGACY

```
<%!  
public class quadrato{  
double lato;  
public quadrato (double L){ lato = L;}  
public double area() {return lato*lato;}  
}  
%>  
....HTML....  
<% quadrato quad1 = new quadrato(request.getParameter("LatoQuadrato")) ;  
double AreaQuad1 = quad1.area();  
%>
```

JSP e LEGACY

- ... quindi posso utilizzare qualsiasi cosa
- primo fra tutti le socket

JSP e SOCKET

- Le socket (con tutti i pro e contro) rappresentano una possibilità di aumentare la granularità di distribuzione del nostro sistema
- nel mondo jsp possiamo utilizzare le socket per comunicare con processi remoti e non.
- nessuno mi vieta di aprire dinamicamente
 - socket
 - serverSocket
 - attenzione all'uso, leggere attentamente le istruzioni del caso !!!!
 - e di recuperare informazioni da queste

JSP e SOCKET

```
• Socket...
• import java.net.*;
import java.io.*;

public class Client {

public static void main (String args[])
/** vuole ip porta arg?
try {Socket s = new Socket (args[0] , Integer.parseInt(args[1]);
System.out.println("connesso");

/** ci costruiamo l'acquisizione e la Socket?
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(is);

OutputStreamWriter os = new OutputStreamWriter(s.getOutputStream());
BufferedWriter bw = new BufferedWriter(os);
/** si scrive vuole argomenti e lunghezza?
String arg = args[1].arg[2].length();
String line;
System.out.println("readLine()");
s.close();
System.out.println("disconnesso");
} catch (IOException e) {
System.out.println("err")
}
}
}
```

JSP e SOCKET

```
• ServerSocket
• import java.net.*;
import java.io.*;

public class Server {

public static void main (String args[])
ServerSocket ss = null;
try {ss = new ServerSocket(8080);
catch (IOException e) {
System.out.println("err");
} while (true)
try {Socket s = ss.accept();
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(is);
OutputStreamWriter os = new OutputStreamWriter(s.getOutputStream());
BufferedWriter bw = new BufferedWriter(os);
String line;
System.out.println("ok");
os.newLine();
os.flush();
s.close();
} catch (IOException e) {
System.out.println("err")
}
}
}
}
```