JSP [1]

Concetti Fondamentali

Slides parzialmente tratte da materiale di Giansalvatore Mecca (*Tecnologie di Sviluppo per il Web*) e Marty Hall (*http://www.coreservlets.com*)

Argomenti della lezione

- Preliminari
  - Architettura
  - Ciclo di Vita
- Sintassi JSP: elementi principali
  - Blocchi di Istruzioni ("Scriptlet")
  - Oggetti Predefiniti
  - Espressioni
  - Dichiarazioni
  - Direttive
- Corrispondenza con i Servlet

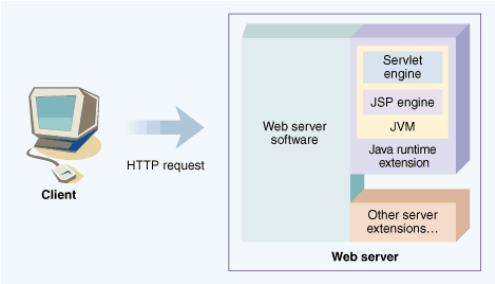
Introduzione

- Java Server Pages (JSP)
- Tecnologia per scripting server-side
  - pagine con codice HTML misto a codice Java
  - proposta da Sun
- Si fonda su tecnologia Java
  - linguaggio Java
  - classi ed interfacce specifiche
    - package javax.servlet.jsp
    - package javax.servlet.jsp.tagext
  - riuso di componenti Java (JavaBean)
    - senza dover necessariamente programmare in Java

Introduzione

- La tecnologia JSP rappresenta un livello costruito sulla tecnologia servlet
  - sintassi rapida per sviluppare Servlet
  - *JSP container* (contenitore) o *JSP engine*
    - programma (es. Tomcat) che compila/esegue JSP
    - parte di un Web application server

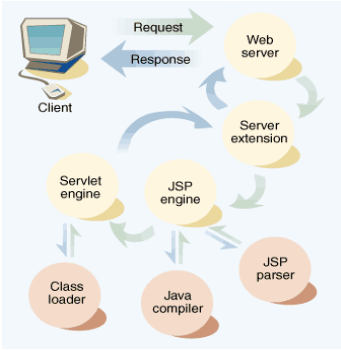
Architettura



Ciclo di vita

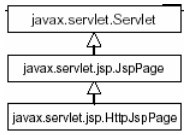
- Il JSP container traduce la JSP in una classe servlet, che gestisce la richiesta corrente e quelle future
- Esecuzione in quattro passi:
  1. il *JSP engine (o JSP container)* parserizza la pagina e crea un sorgente Java (una classe servlet)
  2. il file viene compilato in un file Java *.class*
  3. il contenitore carica e inizializza il servlet
  4. il servlet elabora le richieste e restituisce i risultati
- Note:
  - I passi 1 e 2 vengono eseguiti al primo utilizzo della pagina e ad ogni aggiornamento
  - il passo 3 è eseguito solo alla prima richiesta del servlet
  - il passo 4 può essere eseguito più volte

Ciclo di vita



Ciclo di vita: Servlet generato da una JSP

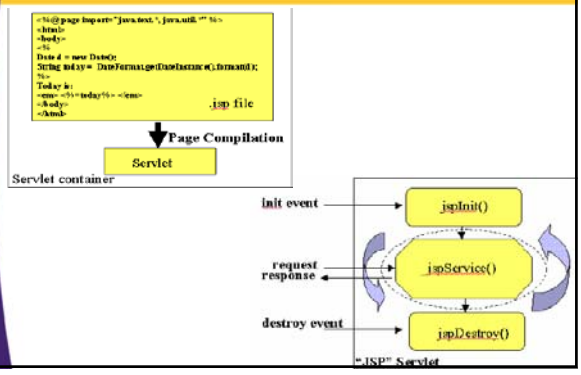
- Si ricompila e inizializza ogni volta che si modifica la JSP
- Estende *javax.servlet.jsp.HttpJspPage*



- Ha un unico metodo di servizio **`_jspService`**
  - chiamato sia per le richieste *get* che per le richieste *post*

```
public void _jspService (HttpServletRequest request,
                         HttpServletResponse response)
```


Ciclo di vita: Servlet generato



Sintassi

Elementi della sintassi JSP

- Blocchi di codice HTML
- Elementi di scripting
  - Commenti
  - Blocchi di istruzioni Java (“scriptlet”)
  - Espressioni
  - Dichiarazioni
- Direttive
  - Messaggi al JSP container (es. Tomcat) che permettono di specificare impostazioni della pagina, inclusione di contenuti da altre risorse, librerie di tag personalizzati
- Azioni
  - Tag predefiniti che incapsulano funzionalità ricorrenti
  - Consentono di creare ed usare oggetti Java (visibili anche nelle scriptlets)

Sintassi: blocchi HTML e commenti

- Blocchi di codice HTML
  - sequenze arbitrarie di codice HTML
  - i tag vengono riscritti identicamente nella risposta
    - generano **`out.print(tag)`** in **`_jspService()`**
- Due tipi di commenti:
  - commenti HTML

    <!-- testo commento -->
  - commenti JSP (non sono riportati nella risposta)

    <%-- testo commento --%>

Scriptlet

- Blocchi di istruzioni Java
  - codice Java utilizzato per costruire la pagina HTML finale
  - permettono di generare dinamicamente porzioni di codice HTML in funzione dell'input
    - analogamente ai servlet
  - appaiono nel metodo *_jspService()* del servlet
- Sintassi: **<% *istruzione Java* %>**
  - si può specificare qualsiasi istruzione ammessa nei metodi *doGet()* o *doPost()* di un servlet
  - nei blocchi è visibile una serie di oggetti predefiniti

Oggetti Predefiniti

Oggetti visibili negli scriptlet (e nelle espressioni)

- `HttpServletRequest` **`request:`**
  - rappresenta la richiesta HTTP
- `HttpServletResponse` **`response:`**
  - rappresenta la risposta HTTP
- `HttpSession` **`session:`**
  - rappresenta la sessione HTTP
- `ServletContext` **`application:`**
  - rappresenta il contesto dell'applicazione
- `JspWriter` **`out`**
  - rappresenta il *PrintWriter* su cui si stampa il corpo della risposta
  - il Content-Type predefinito è "text/html"

Scriptlet:
Codice implicito

- Struttura implicita di _jspService()

```
public void _jspService ( HttpServletRequest request,
                          HttpServletResponse response )
  throws java.io.IOException, ServletException {
        ServletContext application=getServletContext();
        HttpSession session=request.getSession();
        JspWriter out = response.getWriter();
        response.setContentType("text/html");
        ...
}
```

- al corpo di _jspService() si aggiungono:
  - il codice degli scriptlet e delle espressioni
  - le stampe dei tag HTML

Scriptlet:
Esempio

Pagina JSP

```
<html>
<body>
La data di oggi e':
<%
 java.util.Date d =
  new java.util.Date();
 out.print(d);
%>
</body>
</html>
```

Metodo _jspService del servlet

```
public void _jspService (HttpServletRequest
        request, HttpServletResponse response)
throws java.io.IOException, ServletException {
   ServletContext Application=
                getServletContext();
   HttpSession session=
                request.getSession();
   JspWriter out = response.getWriter();
   response.setContentType("text/html");
   out.println("<html>"); out.println("<body>");
   out.println("La data di oggi e':");
   Date d = new java.util.Date();
   out.print(d);
   out.println("</body>"); out.println("</html>");
}
```


Scriptlet:
Esempio

- E' possibile immergere i tag nelle strutture di controllo
  - può diminuire la leggibilità e la mantenibilità della pagina

```
<% if ( Math.random() < 0.5 ) { %>
Have a <B> nice </B> day!
<% } else { %>
Have a <B> lousy </B> day!
<% } %>
```


Espressioni

- Forma rapida per la stampa di stringhe
- Sintassi: <%= *espressioneJava* %>
- Semantica
  - l'espressione è valutata, convertita in String e stampata nella pagina html generata
  - equivalente allo scriptlet:

    <% out.print(espressione.toString()); %>

Esempio: **<%= new java.util.Date() %>** **equivale a**
<% out.print((new java.util.Date()).toString()); %>

Espressioni

- Aumentano la leggibilità del codice
  - Consiglio: ridurre l'uso del metodo `print` nelle pagine JSP ricorrendo ad espressioni equivalenti
- Esempio:

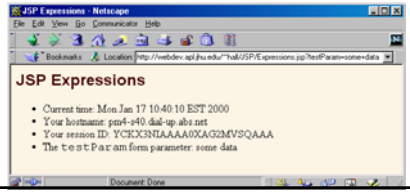
```
<html>
<body>
La data di oggi e':
<%= new java.util.Date() %>
</body>
</html>
```


Alcuni esempi

- Uso di espressioni e scriptlet JSP
- Accesso ai parametri di form

Esempio di uso delle espressioni JSP

```
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY>
```



Oggetto *request*: accesso ad informazioni della richiesta

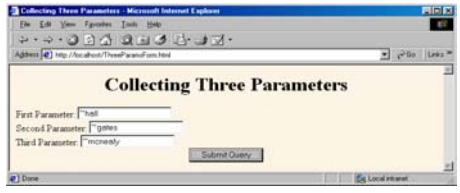
- Accesso ai parametri di form
  - request.getParameter("nome")
    - ritorna il valore della prima occorrenza del parametro *nome* nella query string
    - tratta allo stesso modo richieste GET e POST
    - ritorna *null* se il parametro non esiste
    - ritorna "" se il parametro non ha un valore
  - request.getParameterValues("nome")
    - ritorna un array con i valori di tutte le occorrenze del parametro *nome* nella query string
    - ritorna *null* se il parametro non esiste
  - request.getParameterNames()
    - ritorna una Enumeration di tutti i parametri della richiesta
- Altri metodi
  - request.getMethod(): ritorna il metodo della FORM (GET o POST)
  - request.getHeader("nome"): ritorna il valore di un header

Oggetto *request*: accesso agli header della richiesta

- request.getHeader("nome")
  - Ritorna il valore della prima occorrenza del parametro *nome* nella query string
  - Tratta allo stesso modo richieste GET e POST
  - Ritorna *null* se il parametro non esiste
  - Ritorna "" se il parametro non ha un valore
- request.getParameterValues("nome")
  - Ritorna un array con i valori di tutte le occorrenze di del parametro *nome* nella query string
  - Ritorna *null* se il parametro non esiste
- request.getParameterNames()
  - Ritorna un oggetto *Enumeration* contenente tutti i parametri della richiesta

Esempio2:
Una form HTML con 3 parametri

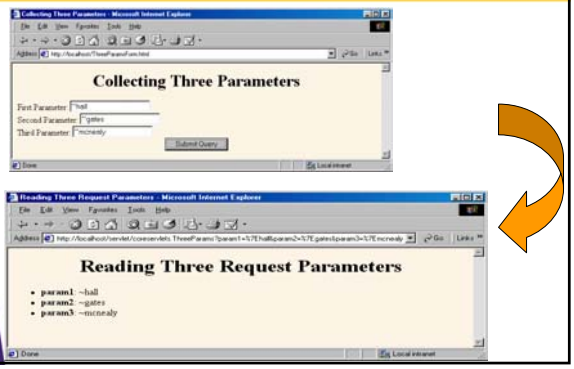
```
<FORM ACTION="/servlet/coreservlets.ThreeParams">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER><INPUT TYPE="SUBMIT"></CENTER>
</FORM>
```



Esempio2:
Stampa di 3 parametri in JSP

```
<HTML>
<BODY>
<H1 ALIGN=CENTER> Reading Three Request Parameters </H1>
<UL>
  <LI><B>param1</B>:
      <%= request.getParameter("param1") %>
  <LI><B>param2</B>:
      <%= request.getParameter("param2") %>
  <LI><B>param3</B>:
      <%= request.getParameter("param3") %>
</UL>
</BODY>
</HTML>
```


Esempio2:
Stampa di 3 parametri in JSP



Dichiarazioni

Dichiarazioni

- Servono a dichiarare elementi esterni a *_jspService*
  - variabili di istanza
    - comuni a tutte le richieste, se un solo oggetto servlet gestisce le varie richieste con thread di _jspService
  - variabili di classe (`static`), comuni a tutte le istanze
  - metodi (di istanza o di classe)
- Attenzione
  - nelle dichiarazioni non sono visibili gli oggetti predefiniti
    - richiesta, risposta, sessione e writer sono inaccessibili
    - sono visibili solo in `_jspService()`
  - soluzione: passarli come parametri ai metodi statici

Dichiarazioni

- In particolare, permettono di ridefinire i metodi
  - jspInit()
    - chiamato da init()
  - jspDestroy()
    - chiamato da destroy()
- operazioni tipiche di jspInit():
  - inizializzazione delle proprietà del servlet
  - inizializzazione del contesto dell'applicazione (ottenibile con `getServletContext()`)
  - creazione di una connessione a database

Dichiarazioni: esempio 1

```
<!DOCTYPE HTML PUBLIC
                "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<HEAD> <TITLE> Random heading </TITLE> </HEAD>
<BODY>
<H1>Some Heading</H1>
<%= randomHeading() %>
```

Dichiarazioni: esempio 2

```
<!DOCTYPE ...>
<HTML>
   <%!
       public String getGreeting (String user) {
         String greeting;
         if (user.equals ("Bob"))
                 greeting = new String ("Hello, Bob!");
         else
                 greeting = new String ("Hello there!");
         return greeting;
       }
   %>
<BODY>
   <H1>
   <%
         String user = request.getParameter ("user");
         if (user == null)  user = new String ("anonymous");
   %>
   <%= getGreeting (user) %>
   </H1>
</BODY>
</HTML>
```

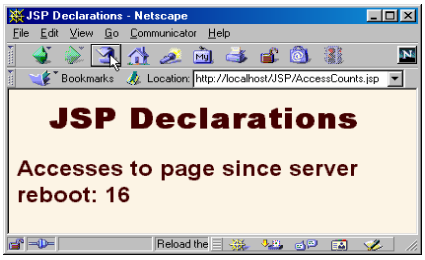

Dichiarazioni: esempio 3

```
<!DOCTYPE HTML PUBLIC
                "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>JSP Declarations</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H1> JSP Declarations </H1>
<%! private int accessCount = 0; %>
<H2> Accesses to page since server reboot:
<%= ++accessCount %> </H2>
</BODY>
</HTML>
```

Dichiararlo *static* se non si è sicuri che le varie richieste siano servite da un solo servlet

Dichiarazioni: esempio 3

Dopo 15 visite effettuate da un numero arbitrario di client, viene visualizzata la pagina seguente:



Direttive

Slides parzialmente tratte da materiale di Giansalvatore Mecca (*Tecnologie di Sviluppo per il Web*) e Marty Hall (*http://www.coreservlets.com*)

Direttive

- Messaggi al JSP container
  - il programma (es. Tomcat) che compila/esegue le JSP
- Permettono al programmatore di specificare:
  - Impostazioni della pagina (direttiva *page*)
  - Inclusione di contenuti da altre risorse (direttiva *include*)
  - Tag personalizzati usati nella pagina (direttiva *taglib*)
- Sintassi:

<%@ direttiva attributo$_1$= "valore$_1$"
…
attributo$_N$= "valore$_N$" %>

Direttiva include

- Direttiva include
  - attributo file
    - *valore: URI di una pagina jsp o html*
  - esempio: **<%@ include file="intestazione.jsp" %>**
  - consente di includere a tempo di traduzione il *codice* della pagina specificata nel servlet generato
    - nella posizione in cui si trova la direttiva include
  - ***ATTENZIONE:*** quando si modifica la pagina inclusa il contenitore NON è obbligato a rigenerare e ricompilare il servlet
- Alternativa: il tag <jsp:include> (vedremo in seguito)
  - *include* a tempo di esecuzione *il* corpo *della risposta prodotta*

Direttiva include

Inclusione di contenuto JSP (*ContactSection.jsp*)

```
<%@ page import="java.util.Date" %>
<%
private int accessCount = 0;
private Date accessDate = new Date();
private String accessHost = "<I> No previous access</I>";
%>
<P>
<HR>
This page &copy; 2000
<A HREF="http//www.my-company.com/">my-company.com</A>.
This page has been accessed <%= ++accessCount %>
times since server reboot. It was last accessed from
<%= accessHost %> at <%= accessDate %>.
<% accessHost = request.getRemoteHost(); %>
<% accessDate = new Date(); %>
```


Direttiva include

Inclusione di contenuto JSP

```
...
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Some Random Page</TABLE>
<P>
Information about our products and services.
<P>
Blah, blah, blah.
<P>
Yadda, yadda, yadda.
<%@ include file="ContactSection.jsp" %>

</BODY>
</HTML>
```


Direttiva include

Inclusione del contenuto JSP: risultato

Direttiva page

- Direttiva ***page***

  <%@ page *attr1="val1", attr2="val2", … %>*

  – attributi principali della direttiva page
    - *import*
    - *errorPage*
    - *isErrorPage*
    - *isThreadSafe*
    - *contentType*
  – vale per tutta la pagina
    - non è posta necessariamente all'inizio

Direttiva page

Importazione di moduli Java

- Attributo ***import*** di page
  – specifica i package e le classi da importare
  – normalmente all'inizio della pagina
  – importazioni automatiche (da non specificare)
    - javax.servlet.*
    - javax.servlet.http.*
    - javax.servlet.jsp.*
- Esempi
  – <%@ page **import**="java.util.Vector" %>
  – <%@ page import="java.util.*" %>
  – importazione delle librerie JDBC:

    <%@ page import="java.sql.*" %>

    nel servlet corrispondente si ha : ***`import java.sql.*;`***

Direttiva page

Importazione di moduli Java

- Nota
  – è opportuno organizzare i componenti in package
    - importare esplicitamente i package con la direttiva page e l'attributo import
  – normalmente sono visibili tutte le classi nella cartella *WEB-INF/classes*
  – non è necessario importare `java.io`
    - non si usa la classse `PrintWriter` ma `JspWriter` (definita in `javax.servlet.jsp`)

Direttiva page

Pagine di errore

- Attributo ***errorPage*** di page
  - specifica l'URI di una pagina (HTML o JSP o Servlet) di errore
    - a cui reindirizzare il browser in caso di errori nell'esecuzione della pagina
- Esempio

```
<%@ page errorPage="errore.jsp" %>
```

  - in caso di errore il browser viene reindirizzato alla pagina "errore.jsp"

Direttiva page

Pagine di errore

- Attributo *iserrorPage* di page
  - Indica se la pagina è utilizzabile come pagina di errore
  - valore *true* o *false* (default)
  - se *true* rende visibile l'oggetto predefinito exception
    - l'eccezione che ha prodotto la chiamata della pagina
- Esempio
  - nella pagina "errore.jsp" deve essere specificato

```
<%@ page isErrorPage="true" %>
```

  - si può così accedere all'oggetto exception, ad es.:

```
<% if(exception!=null)out.print(exception); %>
```


Esempio di pagine di errore: *computeSpeed.jsp*

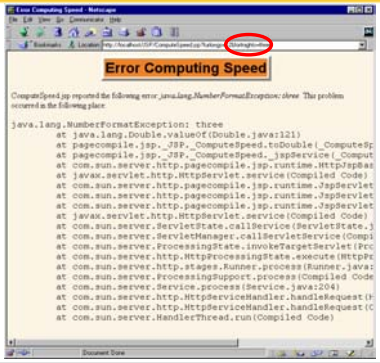
```
...
<BODY>
<%@ page errorPage="SpeedErrors.jsp" %>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE"> Computing Speed </TABLE>
<%! // toDouble non gestisce eccezioni NumberFormatException
private double toDouble(String value) {
   return(Double.valueOf(value).doubleValue());
}
%>
<%
double furlongs = toDouble(request.getParameter("furlongs"));
double fortnights = toDouble(request.getParameter("fortnights"));
double speed = furlongs/fortnights;
%>
<UL>
  <LI>Distance: <%= furlongs %> furlongs.
  <LI>Time: <%= fortnights %> fortnights.
  <LI>Speed: <%= speed %> furlongs per fortnight.
</UL>
...
```


Esempio di pagine di errore: *SpeedErrors.jsp*

```
...
<BODY>
<%@ page isErrorPage="true" %>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      Error Computing Speed</TABLE>
<P>
ComputeSpeed.jsp reported the following error:
<I><%= exception %></I>. This problem occurred in the
following place:
<PRE>
<% exception.printStackTrace(
                  new java.io.PrintWriter(out)); %>
</PRE>
...
```


Esempio di pagine di errore: risposta "normale"

Esempio di pagine di errore: risposta in caso di errore



Direttiva page

Altri attributi

- Attributo session di page
  - controlla l'inizializzazione dell'oggetto `session`
  - possibili valori: true (implicito) | false
    - se è true l'oggetto session viene inizializzato
    - se è false l'oggetto session non viene inizializzato automaticamente (potrebbe non esistere)
- Attributo isThreadSafe di page
  - valore: true (implicito) | false
  - *true*: la pagina non pone problemi di sincronizzazione
    - l'eventuale accesso concorrente deve essere gestito con sezioni critiche (syncronized)
  - *false*: per gestire la sincronizzazione il servlet generato implementa *SingleThreadModel*
    - garantisce che non ci siano due trhead diversi che accedono alla stessa istanza di Servlet contemporaneamente
    - Attenzione: i campi statici potrebbero non essere safe

Esempio di codice Non-Threadsafe

- La pagina JSP assegna ID agli utenti (gli ID devono essere unici!)

```
<%! private int idNum = 0; %>
<%
String userID = "userID" + idNum;
out.println(" Tuo ID=" + userID );
idNum = idNum + 1;
%>
```

- Problema: potrebbero esserci accessi concorrenti
- Soluzioni
  - Impostare *isThreadSafe* = false
  - Definire una sezione critica (soluzione più efficiente)

```
<%! private int idNum = 0; %>
<%
synchronized(this) {
        String userID = "userID" + idNum;
        out.println("Your ID is " + userID + ".");
        idNum = idNum + 1;
}
%>
```


Direttiva page

Attributo contentType

- Attributo contentType di page
  - serve a specificare il formato della risposta come tipo MIME
    - per default si assume una pagina HTML (tipo= text/html)
  - sintassi
    - <%@ page contentType="MIME-Type" %>
    - <%@ page contentType="MIME-Type; charset=Character-Set" %>
  - esempi:

```
<%@ page contentType="text/plain" %>
<%@ page contentType="application/vnd.ms-excel" %>
```

  - dall'header *accept* della richiesta è possibile ricavare quali sono i formati accettati dal client

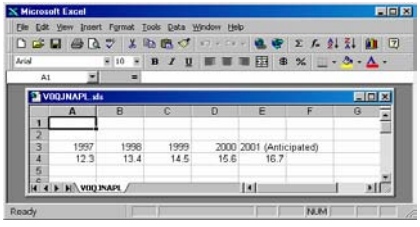
```
request.getHeader("Accept")
```


Principali tipi MIME

application/msword	documento Microsoft Word
application/octet-stream	dati binary
application/pdf	file Acrobat (.pdf)
application/postscript	file PostScript
application/vnd.ms-excel	foglio Excel
application/vnd.ms-powerpoint	presentazione Powerpoint
application/x-gzip	archivio Gzip
application/x-java-archive	file JAR
application/x-java-vm	file bytecode Java (.class)
application/zip	archivio Zip
audio/basic	file musicale .au o .snd
audio/x-aiff	file musicale AIFF
audio/x-wav	file musicale Windows
audio/midi	file musicale MIDI
text/css	foglio di stile HTML
text/html	documento HTML
text/plain	file di testo
text/xml	documento XML
image/gif	immagine GIF
image/jpeg	immagine JPEG
image/png	immagine PNG
image/tiff	immagine TIFF
video/mpeg	video clip MPEG
video/quicktime	video clip QuickTime

Esempio d'uso di contentType: visualizzazione di un foglio Excel

```
<%@ page contentType="application/vnd.ms-excel" %>
<%-- NB: le colonne sono separate da tab e non da spazi --%>
1997    1998    1999    2000    2001
12.3    13.4    14.5    15.6    16.7
```



Visualizzazione condizionale di un foglio Excel

- *Idea:* scelta del formato della risposta sulla base dei parametri della richiesta
  - Excel può interpretare tabelle HTML
- *Problema:* nella direttiva page non è possibile inserire valori dinamici
- *Soluzione:* uso del metodo *setContentType* dell'oggetto implicito *response*

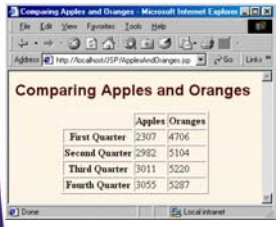
```
<%
if (someCondition) {
  response.setContentType("type1");
} else {
  response.setContentType("type2");
}
%>
```

Visualizzazione condizionale di un foglio Excel

```
<%
String format = request.getParameter("format");
if ((format != null) && (format.equals("excel"))) {
  response.setContentType("application/vnd.ms-excel");
}
%>
<!DOCTYPE ...>
<HTML><HEAD> <TITLE>Comparing Apples and Oranges</TITLE> </HEAD>
<BODY> <CENTER>
<H2>Comparing Apples and Oranges</H2>
<TABLE BORDER=1>
  <TR><TH></TH><TH>Apples<TH>Oranges
  <TR><TH>First Quarter<TD>2307<TD>4706
  <TR><TH>Second Quarter<TD>2982<TD>5104
  <TR><TH>Third Quarter<TD>3011<TD>5220
  <TR><TH>Fourth Quarter<TD>3055<TD>5287
</TABLE>
</CENTER> </BODY>
</HTML>
```


Visualizzazione condizionale di un foglio Excel: risultato

Formato html (default)



Foglio Excel

Sommario

- Sintassi JSP – elementi principali:
  - blocchi di codice HTML
  - commenti:
    <%-- *commento* --%>
  - scriptlet:
    <% *istruzione Java* %>
  - espressioni:
    <%= *espr. Java* %>
  - dichiarazioni:
    <%! *dich. Java* %>
  - direttive:
    <%@ *direttiva* %>

Sommario:
Corrispondenza con le Servlet

- Pagine JSP
  - Appaiono come HTML or XHTML
  - Utilizzate soprattutto quando gran parte del contenuto è costituito da fixed-template data
- Servlets
  - Utilizzati quando il contenuto è quasi tutto generato dinamicamente
  - Alcuni servlet non producono contenuti
    - Invocano altri servlet e pagine JSP

Sommario:
Corrispondenza con le Servlet

- Le JSP sono eseguite come parte di un Web server
  - JSP container
  - Il JSP container traduce la JSP in una servlet
    - Gestisce la richiesta corrente e quelle future
- Errori per le pagine JSP
  - Translation-time
    - Durante la traduzione della pagina JSP in servlet
  - Request-time
    - Durante l'elaborazione della richiesta

Sommario:
Corrispondenza con le Servlet

- la pagina JSP viene tradotta in una classe servlet dal contenitore
  - è una sottoclasse di servlet
    - javax.servlet.jsp.HttpJspPage
  - metodi fondamentali
    - _jspService()
      - *Traduzione del contenuto della pagina*
      - *Usato per qualunque tipo di richiesta (GET, POST)*
    - jspInit() e jspDestroy()
      - *Invocati dal container all'inizializzazione e alla terminazione*
  - altri metodi
    - si possono definire con opportuni script (dichiarazioni)
- òa servlet viene ricompilato e re-inizializzato ogni volta che il file jsp viene modificato

Sommario:

Corrispondenza con i Servlet

- Costruzione del servlet
  1. vengono importati i package standard e quelli specificati nella direttiva page
  2. vengono incluse le proprietà ed i metodi definiti nelle dichiarazioni
  3. vengono inclusi i commenti Java
  4. nel metodo *_jspService()* vengono inizializzati gli oggetti predefiniti
  5. viene assegnato il *Content-Type* predefinito
  6. i tag della pagina *jsp* danno origine a chiamate out.print("*tag*");
  7. il codice degli scriptlet viene incluso ordinatamente nel metodo *_jspService()*

Esempio 2

Stampa di tutti i parametri

```
... <BODY>
<H1 ALIGN=CENTER>"Reading all parameters"</H1>
<TABLE BORDER=1 ALIGN=CENTER>
<TR> <TH>Parameter Name<TH>Parameter Value(s)
<% Enumeration paramNames = request.getParameterNames();
   while(paramNames.hasMoreElements()) { %>
        <TR><TD> <%= paramNames.nextElement() %> <TD>
        <% String[] paramValues = request.getParameterValues(paramName);
        if (paramValues.length == 1) {
                String paramValue = paramValues[0];
                if (paramValue.length() == 0) out.println("<I>No Value</I>");
                else                          out.println(paramValue);
        }
        else {   out.println("<UL>");
                 for(int i=0; i<paramValues.length; i++) {
                                out.println("<LI>" + paramValues[i]);
                 }
                 out.println("</UL>");
        }
} %>
</TABLE>
</BODY></HTML>
```