Java Server Page

🤔 **DISCLAIMER!**

In questa lezione ho avuto un sacco di problemi tecnici.

Un sito che racchiude meglio i contenuti principali da sapere per utilizzare le JSP è questo:

https://www.w3schools.in/jsp/tutorials/

Codice status della response di un Server

La risposta nel modello client-server, dalla parte del server può restituire uno **status code**.

Lo status code è un numero che rappresenta “il risultato” della richiesta. Questi sono:

1. **200, “successo”** → Il server ha ricevuto la richiesta correttamente, e sta restituendo correttamente i dati.
2. **400 “errore”** → Il server non dispone del dato / della risorsa richiesta, si verifica un errore.

Ci sono altri status code, ma questi sono i “principali” messaggi in una conversazione client-server.

Jsp (Java Server Page)

E’ una tecnologia che permette di realizzare delle semplici applicazioni web.

Viene iniettato del codice Java all’interno di una pagina HTML, tramite un set di metodi.

E’ un’estensione delle Servlet.

Vantaggi e svantaggi di JSP

Vantaggi

- Facile da mantenere
- Abbastanza veloce (da imparare ed implementare)

Svantaggi

- La compilazione di una JSP è abbastanza oneroso in termini di prestazioni.

Elementi di scripting

Sono degli elementi, di JSP, che permettono di iniettare del codice Java dentro le Java Server Page.

Gli elementi di scripting si suddividono in 3 tipi:

1. **Scriplet**
   → Per il codice java generico.
2. **Expression**
   → Per portare in output qualcosa (evitare di scrivere out.print()).
3. **Declaration**
   → Dichiarazione di attributi e metodi.

Scriplet

Lo scriplet è una notazione che mi permette di scrivere codice Java internamente a dei tag HTML.

```
<body>
  <% posso inserirci codice java %>
</body>
```

Expression tag

Permette di stampare lo stream di output di una funzione,

la sintassi è:

```
<%=  statement %>

html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Dichiarazione

Serve per dichiarare delle nuove variabili nel mezzo del codice html, la sintassi è:

```
<html>
    <body>
    <%! int data=50; %>
    <%= "Value of the variable is:"+data %>
    </body>
  </html>
```

Setting di Eclipse

Dal marketplace → installare “**Eclipse Enterprise Java and Web Developer tools**” dal **help > marketplace.**

Successivamente installare **Tomcat** e collegarlo ad **Eclipse** (molto complicato, non riesco a ricordare come l’ho fatto).

Creazione di un progetto JSP

Per creare il progetto:

File > new File > create Dynamic Web App

Poi:

Copiare l’html creato ieri in progetto > src > main > webapp,

E devo creare una JSP che **conterrà i risultati** dell’ interrogazione al DB.

Deploy dell’applicazione

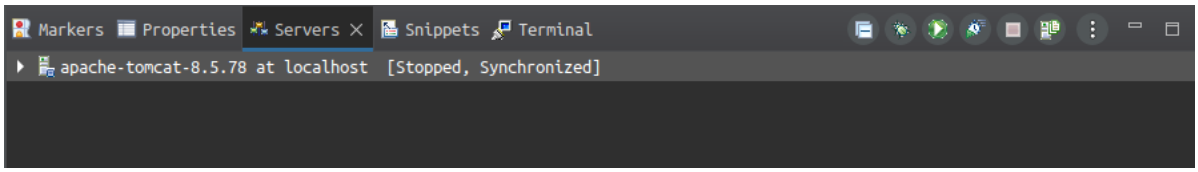
> 🤔 **Errori comuni!**
>
> Ogni tanto capita di avere già la porta 8080 occupata (ad esempio io nell’esercizio precedente avevo ancora attivo xampp).
>
> Bisogna sempre lasciarla libera, perchè questa è proprio la porta utilizzata da Tomcat. **Assicurarsi di avere sempre la porta libera prima di avviare il server Tomcat.**

Ora, **una volta che ho il progetto ben configurato**, bisogna fare due operazioni:

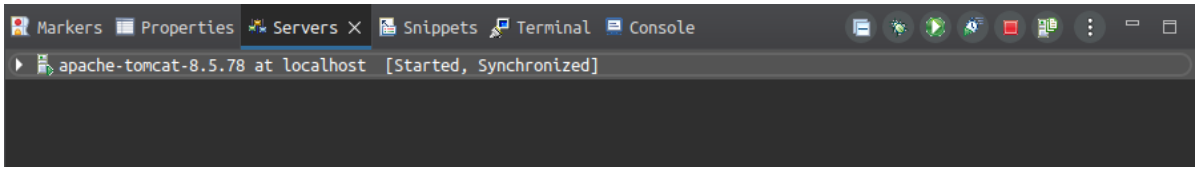
1. **Startare il server**

Fare click destro sul nome del server nella console di Eclipse).
Il risultato deve essere “Started, Synchronized” alla fine.

Inizio:



Dopo click destro e “start”:



2. **Deployare la nostra app sul server**

Ora, per “fare partire la nostra applicazione sul server”, basta:

Fare click destro sul codice html della pagina (su Eclipse) e Poi run As > Run on Server.

Ci esce una finestra → Selezioniamo “Tomcat” come server → e facendo finish dovebbe **aprirsi localhost:8080 con la nostra applicazione dentro.**

Oggetti della Server Page

Alla creazione di una JSP vengono automaticamente creati 9 oggetti per la gestione di informazioni Client-Server.

A noi in questo caso servono solo 2 di questi oggetti: request e response.

JSP request

E’ l’oggetto che prende le informazioni di richiesta da un form.

JSP Response

E’ l’ oggetto di “risposta” ad una richiesta

Prendere oggetti dal form e portarli nella Server Page

Se nel form ho questi input:

```
<input type="text" name="Targa" value="" placeholder="Inserisci la targa dell'auto">
<input type="text" name="Tipologia" value="" id="tipologia" placeholder="Inserisci la tipologia dell'auto">
<input type="text" name="Produttore" value=""  id="produttore" placeholder="Inserisci il Produttore dell'auto">
<input type="submit" class="pulsante" name="" value="Invia Richiesta">
```

Allora nella **JSP**, per prendere i parametri richiesti dal form:

```
//Nella Java Server Page:

<%

  // Prendiamoci i parametri dalla richiesta del form
  String Targa = request.getParameter("Targa");
  String Tipologia  = request.getParameter("Tipologia");
  String Produttore = request.getParameter("Produttore");

  out.println(Targa + " " + Tipologia + " " + Produttore);

%>
```

Response.sendRedirect();

Questa funzione dell’ oggetto Response permette di reinderizzare l’applicazione verso un altro sito.

```
<%
```

```
  // ESEMPIO: questo file è applicazione.JSP
  // Prende le informazioni da un form che vive in index.html

  // Stampa tali informazioni e subito dopo, reindirizza l'utente
  // Verso una pagina HTML che dice "Hai inserito tutto con successo"

  // Prendiamoci i parametri dalla richiesta del form
  String Targa = request.getParameter("Targa");
  String Tipologia  = request.getParameter("Tipologia");
  String Produttore = request.getParameter("Produttore");

  out.println(Targa + " " + Tipologia + " " + Produttore);

  // Return alla pagina principale
  response.sendRedirect("<http://localhost:8080/Garage/inserimento_avvenuto.html>");
%>
```

Direttiva include

Posso includere un file in un altro con la direttiva:

```
// Direttiva dentro un filex.jsp
<%@ include file = "header.html">
```

Cosa fa questo codice?

Prende il contenuto di header.html e lo copia direttamente **dentro la JSP.**

Action element

Sono delle funzioni delle Servlet, che vengono invocate.

Quelle che ci interessano sono:

forward

Vogliamo passare da una JSP ad un’altra JSP. Usiamo:

```
<jsp.forward page ="url relativo" />
```

param

Andare ad una nuova pagina, passando un parametro inserito dall’utente:

```
<jsp.forward page = "url relativo">

  // Qua sto passando un valore dalla jsp corrente alla successiva
  <jsp.param name="name" value="valore passato" />

</jsp.forward>
```

java Bean

E’ una classe di Java con un “formato” particolare:

1. Non ha un costruttore
2. Implementa il serializable
3. Deve implementare set e get **per ogni attributo**

Dalla nostra classe Garage ad esempio:

→ basta togliere il costruttore e mettere **set e get** per ogni attributo della classe.

Per creare ed utilizzare un oggetto Bean:

```
<jsp:useBean id="nome_oggetto" class="nome_classe"/>

// che vuol dire: sto istanziando un oggetto del tipo "nome_classe" e gli sto
// dando l'id (un identificativo) "nome_oggetto"
```

setProperty

Serve per invocare la set di un attributo:

```
<jsp.setProperty property="elenco_proprietà" name="oggetto_bean"  />
// Ossia -> voglio settare un nuovo valore per questo "elenco_proprietà"
// per l'oggetto chiamato "oggetto_bean"

//OCCHIO -> il contenuto che sto passando lo prendo da un FORM
```

getProperty

Invoca la get di un oggetto Bean:

```
<jsp.getProperty property="attributo" name ="nome_oggetto"/>
```

Esempio esplicativo:

https://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag

---

Esercizio!

Utilizziamo la classe Garage → ed utilizziamo come classe Bean per fare il controllo del Database con le JSP.

> https://s3-us-west-2.amazonaws.com/secure.notion-static.com/44a6427b-450f-4161-9921-a3074dc1192c/GarageBean(funziona!!).zip

---

Prima del deploy

Ricordarsi di importare il Jconnector tra le librerie del progetto.

Incollare il Jconnector **anche dentro la cartella tomcat/lib** (dipende da dove abbiamo installato tomcat.

Ricordarsi di avviare il Database Mysql con xampp
