Lezione 10 - JDBC

Azioni standard

`<jsp:include page="url" />`: Per includere risorse staticamente oppure dinamicamente al request time.

`<jsp:forward page="url" />`: Per inoltrare al client la pagina specificata nell'url (la jsp che richiama questa azione non deve aver inviato output, altrimenti otteniamo un'eccezione del tipo IllegalStateException;

`<jsp:param name="prezzo" value="34" />`: Per specificare parametri da passare alla pagina inclusa o inoltrata (vedi 2 azioni precedenti);

`<jsp:plugin type="applet" code="Molecule.class" codebase="/html" >`: per includere i tag per la gestione delle Applet (vedremo più avanti).

Esempi:

```
<jsp:forward page="urlSpec">
  <jsp:param name="nome"
value="valore"/>
</jsp:forward>
```

In una servlet possiamo "inoltrare" la generazione della pagina HTML alla pagina JSP con il seguente comando:

```
ServletContext sc =
getServletContext();
RequestDispatcher rd =
sc.getRequestDispatcher(jsp);
rd.forward(req,res);
```

Nella servlet possiamo usare il comando visto prima solo se prima non abbiamo mai scritto sull'oggetto response. Nella pagina JSP è più difficile controllare se è stato scritto qualcosa, per questa ragione il motore delle servlet mantiene un buffer con i caratteri scritti nel out. Finché i caratteri scritti nel buffer non vengono inviati effettivamente al client tramite il socket, è possibile usare jsp:forward. Eventualmente è possibile agire sulla grandezza del buffer con la direttiva “buffer”.

Esempio uso applet:

```
<jsp:plugin type="applet" code="Molecule.class"
codebase="/html" >
  <jsp:params>
    <jsp:param name="molecule"
value="molecules/benzene.mol"/>
  </jsp:params>
  <jsp:fallback>
    <p> unable to start plugin </p>
  </jsp:fallback>
</jsp:plugin>
```

Differenza tra direttiva include e azione include

Tipo	Sintassi	File o Pagina inclusa	Descrizione	Tempo esecuzione
Include Directive	`<%@ include file=... %>`	Statica	Contenuto tradotto dal motore delle servlet	Traduzione
Include Action	`<jsp:include page= />`	Statica o dinamica	Contenuto incluso tale e quale (senza traduzione)	Esecuzione

Esercizi

1. Collegare una servlet e una JSP in modo che (parte del)l'elaborazione venga fatta dalla servlet mentre la creazione della pagina html dalla jsp.

2. Usare la direttiva include per includere l'intestazione di un sito in più pagine jsp;

3. Usare l'azione include per visualizzare nella pagina "`login.jsp`", la pagina "`logged.jsp`" in caso di successo nell'autenticazione, e "`loginError.jsp`" in caso di insuccesso. Per semplicità assumete che tutte le stringhe `user` e `password` vadano bene se e solo se "`user.equals(password)`".

JDBC

Nella maggior parte delle applicazioni reali si ha bisogno di accedere a un database. Per questo motivo è stata introdotta la Java DataBase Connectivity (JDBC) libreria per accedere ai database.

Per poter accedere a differenti tipi di DataBase, JDBC è stata suddivisa in due parti: 1) l'interfaccia standard uguale per tutti i DB e 2) il driver che implementa l'interfaccia che dipende dal DB. Il driver è una libreria di norma contenuta in un file .jar.

Per accedere ad un DB dobbiamo quindi aggiungere al classpath dell'applicazione il driver JDBC opportuno. Per le servlet possiamo mettere il driver nella cartella lib dell'applicazione (e in questo modo ogni applicazione che usa JDBC anche con lo stesso DB dovrà avere la propria copia della libreria) oppure nella cartella lib del motore delle servlet (in questo modo la libreria è condivisa tra tutte le applicazione del motore delle servlet).

Il fatto che il driver sia separato dall'interfaccia permette di poter cambiare il tipo di DB (per esempio passando da Oracle a DB2) senza bisogno di ricompilare l'applicazione ma solamente cambiano il driver.

Dal punto di vista della programmazione per accedere ai dati di un DB con questa libreria dobbiamo fare tre cose:

1. stabilire una **connessione** con la sorgente dei dati;
2. inviare **comandi** di interrogazione e modifica dei dati;
3. elaborare i **risultati**.

Per ognuna delle operazioni precedenti è presente una classe JDBC incaricata della gestione delle operazioni relative alle funzionalità della stessa:

1. `Connection` per collegarsi al DB;
2. `Statement` per eseguire query SQL al "volo" o PreparedStatement per SQL dinamico;
3. ResultSet racchiude il risultato di una query.

Il ResutlSet è sostanzialmente un array bidimensionale: di righe e colonne. Per navigare tra le colonne possiamo usare l'eccesso tramite la posizione della colonna, la colonna con posizione 1 è la prima (attenzione che gli array in java partono da 0 invece che 1) o tramite il nome della colonna se esiste. Mentre per navigare tra le righe possiamo usare uno dei metodi che sposta il cursore di una certa posizione:

- `next()` per passare alla prossima riga (indice ++);
- `previous()` per tornare alla riga precedente(indice--);
- `last()` per passare all'ultima riga(indice=size);
- `first()` per andare alla prima riga(indice=1);
- `absolute(int pos)` per andare alla riga pos(indice=pos);

Attenzione che non tutti i `ResultSet` sono "scrollabili" all'indietro o in maniera assoluta, quando si crea un `ResultSet` ci sono dei parametri da specificare per chiedere esplicitamente di poter navigare in tutte le direzioni.

tabella: amici	a	b	c	
1	23	"Alberto"	11.23	first()
2	43	"Bianca"	12.34	
3	33	"Evelina"	-12.05	
4	11	"Bruno"	3.23	absolute(4)
5	-12	"Nicola"	22.22	
6	17	"Sandro"	56.00	
7	10	"Valerio"	32.12	
8	11	"Matteo"	00.00	
9	12	"Paola"	23.89	last()

Nel seguente esempio vediamo come interagiscono le tre classi in questione:

```
  Connection con =
DriverManager.getConnection("jdbc:myDriver:wombat","myLogin", "myPassword");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM amici");
  while (rs.next()) //per passare alla prossima riga
  {
    int x = rs.getInt("a"); // o int x = rs.getInt(1);
    String s = rs.getString("b"); //o String s = rs.getString(2);
    float f = rs.getFloat("c"); //o float f = rs.getFloat(3);
  }
  rs.close();
  stmt.close();
  con.close();
```

I comandi sono espressi tramite un comando SQL. Alternativamente agli `Statement` che vengono creati al volo e usati una volta solo, esiste un'altro tipo di comandi, detto `PreparedStatement`.

I `PreparedStatement` sono da preferire quando:

1. abbiamo delle istruzioni SQL con parametri (gestione in automatico da parte del driver JDBC della formattazione da tipi Java a tipi del DB ad esempio le date e le stringhe);
2. velocizzare le query ripetute;
3. sicurezza sulla esatta sintassi della query SQL risultante: evita i problemi di "SQL Injection" (http://en.wikipedia.org/wiki/SQL_injection).

Ecco un esempio d'uso di un prepared statement:

```
PreparedStatement ps t= conn.prepareStatement("SELECT * FROM Tab WHERE
s= ? ");
String nome = req.getParameter("nome");
pst.setString(1,nome);
ResultSet rs = stmt.executeQuery();
while (rs.next())
{
   int x = rs.getInt("a");
}
```

Un esempio completo di Servlet che accede a un DB MicrosoftAccess tramite bridge JDBC ODBC è il seguente:

```
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import interbase.interclient.*;

public class Leggi extends HttpServlet {
    /** Processes requests for both HTTP <code>GET</code> and
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet Leggi</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("<ul>");

try{
          Class driver = Class.forName("sun.jdbc.odbc.JdbcOdbcConnection");
          java.sql.Connection c = java.sql.DriverManager.getConnection("jdbc:odbc:nome");
          java.sql.Statement s = c.createStatement();
          java.sql.ResultSet rs = s.executeQuery("SELECT * FROM dati");

          while (rs.next()){
            out.println("<li>"+rs.getString("descrizione"));
          }
        }
        catch (Exception e )
        {
```

```
            e.printStackTrace();
        }

        out.println("</ul>");
               out.println("</body>");
        out.println("</html>");
        out.close();
    }

    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }

}
```

Esempio codice accesso a un Database

In questo semplice esempio possiamo vedere come accedere al database Microsoft Access di cui abbiamo creato un DNS ("nome" nell'esempio di sopra) che collega il DB vero e proprio all'applicazione Java.
nota: db url, nome utente e password per accedere al database andrebbero inseriti tra i parametri di configurazione della servlet o della web-application per essere letti e impostati nel metodo `init` della servlet stessa in questo modo è possibile cambiare DB, user e password senza bisogno di ricompilare l'applicazione, inoltre il programmatore non viene a conoscenza delle credenziali di accesso al DB.

Un esempio di inserimento dei dati di un DB è il seguente:

```
try{
  Class driver = Class.forName("sun.jdbc.odbc.JdbcOdbcConnection");
  java.sql.Connection c = java.sql.DriverManager.getConnection("jdbc:odbc:nome");
  preparedStatement =
     connection.prepareStatement("INSERT  INTO commenti (login,commento) VALUES (?,?)");

  preparedStatement.setInt(1,request.getParameter("LOGIN"));
  preparedStatement.setString(2,Integer.parseInt(request.getParameter("Commento")));
  int res = preparedStatement.executeUpdate();
  preparedStatement.close();
  connection.close();
  //Inserite  res righe
}
catch (Exception e )
{
```

```
      e.printStackTrace();
}
```

Esempio inserimento dati del Database

In questo esempio aggiungiamo una nuova riga al database tramite il comando SQL INSERT.

Altri comandi SQL molto usati sono DELETE per cancellare una riga e l'UPDATE per modificare i valori di una riga.
Un esempio di modifica di una riga in un DB è il seguente:

```
try{
     Class driver = Class.forName("sun.jdbc.odbc.JdbcOdbcConnection");
     java.sql.Connection c = java.sql.DriverManager.getConnection("jdbc:odbc:nome");
     preparedStatement =
          connection.prepareStatement("UPDATE  commenti SET commento=? WHERE login=?)");

     preparedStatement.setInt(2,request.getParameter("LOGIN"));
     preparedStatement.setString(1,Integer.parseInt(request.getParameter("Commento")));
     int res = preparedStatement.executeUpdate();
     preparedStatement.close();
      connection.close();
     //Modificate  res righe
}
catch (Exception e )
{
      e.printStackTrace();
}
```

Esempio modifica dati del Database

Nell'esempio vengono modificate tutte le righe con un dato login: in particolare viene impostata la colonna commento con il valore del parametro "Commento" della richiesta.

Un esempio di cancellazione di righe da un DB è il seguente:

```
try{
     Class driver = Class.forName("sun.jdbc.odbc.JdbcOdbcConnection");
     java.sql.Connection c = java.sql.DriverManager.getConnection("jdbc:odbc:nome");
     preparedStatement =  connection.prepareStatement("DELETE FROM commenti where login=?"

     preparedStatement.setInt(1,request.getParameter("LOGIN"));
     int res = preparedStatement.executeUpdate();
     preparedStatement.close();
     connection.close();
     //Cancellate  res righe
}
catch (Exception e )
{
      e.printStackTrace();
}
```

Esempio di cancellazione dati del Database

In questo esempio vengono cancellate tutte le righe di un certo login.