

Gestione di immagini/file in PostgreSQL e servlet

Gabriele Pozzani

Laboratorio di Basi di Dati
Laurea in Informatica

25 maggio 2010

Sommario

- 1 Approcci
- 2 Oggetti fondamentali
 - PostgreSQL
 - Form
 - Classi JAVA
- 3 Web Application
 - Nozioni preliminari
 - Servlet
 - DBMS
 - Visualizzazione
- 4 Installare l'esempio

Sommario

- 1 Approcci
- 2 Oggetti fondamentali
 - PostgreSQL
 - Form
 - Classi JAVA
- 3 Web Application
 - Nozioni preliminari
 - Servlet
 - DBMS
 - Visualizzazione
- 4 Installare l'esempio

Obiettivo

Nostro obiettivo è...

... vedere come poter memorizzare e gestire file, e in particolare immagini, tramite PostgreSQL e una Web Application.

Oggi vedremo come gestire immagini, ma le stesse idee possono essere facilmente estese a qualunque tipo di file.

Possibili approcci

Vi sono due possibili approcci che possono essere seguiti:

- 1 memorizzare nel DB direttamente le immagini;
- 2 salvare le immagini sul disco e memorizzare nel DB i loro path.

Vedremo entrambi gli approcci tramite l'esempio di una semplice Web Application che:

- memorizza nome e cognome di una persona;
- ad ogni persona associa una foto/immagine.

Sommario

- 1 Approcci
- 2 **Oggetti fondamentali**
 - PostgreSQL
 - Form
 - Classi JAVA
- 3 Web Application
 - Nozioni preliminari
 - Servlet
 - DBMS
 - Visualizzazione
- 4 Installare l'esempio

Tipo di dato `bytea`

Al fine di memorizzare file, immagini e video in PostgreSQL:

- possiamo utilizzare il tipo di dato `bytea`;
- `bytea` permette di memorizzare stringhe binarie, cioè sequenze di byte;
- le stringhe binarie si distinguono dalle stringhe di caratteri perché
 - consentono di codificare anche valori che non sono ammessi dalla codifica dei caratteri scelta per il DB.
 - le operazioni sono operazioni generiche su byte e non dipendono dalla codifica scelta per i caratteri.

multipart/form-data content type

Il tag HTML `FORM` possiede l'attributo `enctype`. Esso permette di specificare la codifica dei valori da trasmettere alla pressione del tasto submit.

Normalmente i parametri di una richiesta HTTP vengono codificati usando ASCII e usando i caratteri di escape per i caratteri riservati.

Tale codifica è identificata da

`"application/x-www-form-urlencoded"`.

Tale codifica è inefficiente per trasmettere grandi quantità di dati, come i file. A tal fine è possibile usare la codifica

`"multipart/form-data"`.

Questa codifica è stata definita dall'Internet Engineering Task Force (IETF):

`http://www.ietf.org/rfc/rfc1867.txt`

Input FILE

Insieme al nuovo tipo di codifica è stata anche definito il nuovo tipo FILE per gli INPUT delle FORM.

```
<INPUT TYPE="FILE" NAME="IMAGE" SIZE=35>
```

Quando selezionato esso permette di scegliere un file dal disco.

Il file selezionato viene poi codificato e inviato tramite la codifica `multipart/form-data`.

Esempio

```
<FORM METHOD="POST" ACTION=...  
  ENCTYPE="multipart/form-data">  
...  
<INPUT TYPE="FILE" NAME="IMAGE" SIZE="35">  
...  
</FORM>
```

ENCTYPE

Bisogna specificare che i dati della form verranno inviati tramite la codifica `multipart/form-data`.

Esempio

```
<FORM METHOD="POST" ACTION=...  
  ENCTYPE="multipart/form-data">  
...  
<INPUT TYPE="FILE" NAME="IMAGE" SIZE="35">  
...  
</FORM>
```

TYPE

Il tipo di input FILE è simile agli altri tipi di input ma permette di specificare un file.

Esempio

```
<FORM METHOD="POST" ACTION=...  
  ENCTYPE="multipart/form-data">  
...  
<INPUT TYPE="FILE" NAME="IMAGE" SIZE="35">  
...  
</FORM>
```

METHOD

Dovendo inviare grandi quantità di dati non è possibile usare il metodo GET. Dobbiamo usare il metodo POST che invia i dati "in background".

Libreria `cos.jar`

In **JAVA** la gestione di contenuti, tra cui file e immagini, codificati con `multipart/form-data` si avvale della libreria `jar cos`

```
http://www.servlets.com/cos/
```

Al suo interno la classe più importante è `com.oreilly.servlet.MultipartRequest`

All'interno del metodo `doPost` di una servlet un oggetto `MultipartRequest` può essere ottenuto dall'oggetto `HttpServletRequest`

```
MultipartRequest multi;  
multi = new MultipartRequest(request, "/tmp/");
```

Il secondo parametro specifica dove salvare temporaneamente eventuali file.

MultipartRequest

Da una variabile di tipo `MultipartRequest` è possibile recuperare eventuali parametri della servlet, similmente a come si fa con oggetti `HttpServletRequest`, usando il metodo `getParameter()`

```
String par;  
par = (String)multi.getParameter(nomeParametro);
```

Nel caso di file si utilizza il metodo `getFile()` che restituisce un oggetto di tipo `File` che punta al file “temporaneo” salvato nella directory specificata prima.

```
File f = multi.getFile(nomeParametro);
```

È poi possibile operare come si vuole sul file.

Tipi di file

È possibile controllare il tipo di un file verificandone il tipo MIME.

```
import javax.activation.MimetypesFileTypeMap;
...
String mimetype;
mimetype=(new MimetypesFileTypeMap()).getContentType(f);
if (mimetype.equals("image/jpeg")) {
    THEN
} else {
    ELSE
}
```

<http://en.wikipedia.org/wiki/MIME>

http://www.w3schools.com/media/media_mimeref.asp

FILE

In JAVA si possono leggere e scrivere files tramite le classi `FileInputStream` e `FileOutputStream`.

```
File fIN = new File(filepathIN);
File fOUT = new File(filepathOUT);
//Apro i file stream in ingresso (da cui leggere
//il file originale)...
FileInputStream fIS = new FileInputStream(fIN);
//...e in uscita (su cui scrivere l'immagine)
FileOutputStream fOS = new FileOutputStream(fOUT);
//copio byte per byte l'immagine dallo stream
//in ingresso a quello in uscita
while (fIS.available()>0)
    fOS.write(fIS.read());
//chiudo gli stream
fIS.close();
fOS.close();
```

Sommario

- 1 Approcci
- 2 Oggetti fondamentali
 - PostgreSQL
 - Form
 - Classi JAVA
- 3 **Web Application**
 - Nozioni preliminari
 - Servlet
 - DBMS
 - Visualizzazione
- 4 Installare l'esempio

Funzionalità

La Web Application di esempio permette di:

- inserire nel DB un nuovo record impostando nome e cognome della persona e l'immagine da associarvi
 - un check box nella form di inserimento dei dati permette di scegliere se
 - checkbox selezionato: l'immagine va memorizzata direttamente nella tabella;
 - checkbox deselezionato: l'immagine va salvata in un'apposita cartella per poi inserirne il path nel DB.
- recuperare i record nella tabella tramite un'apposita form di ricerca in cui scegliere nome e/o cognome o nessuno dei due (per ottenere tutte le tuple).

Struttura

La Web Application è composta da:

- quattro JSP per la presentazione di risposte, risultati, ecc. . .
- una servlet centrale (`photos`), che riceve tutte le richieste, esegue le operazioni richieste e richiama la JSP per la presentazione dei risultati
- una classe `DBMS` (più eventuali bean) che gestisce l'interazione tra `photos` e il database

La servlet sceglie quale operazione eseguire osservando il valore di un apposito parametro, `command`.

DB di riferimento

La Web Application si basa sulla tabella PeoplePicture:

Colonna	Tipo	Proprietà
id	serial	primary key
name	varchar(30)	not null
surname	varchar(30)	not null
picturepath	varchar(128)	
picture	bytea	

Vincoli:

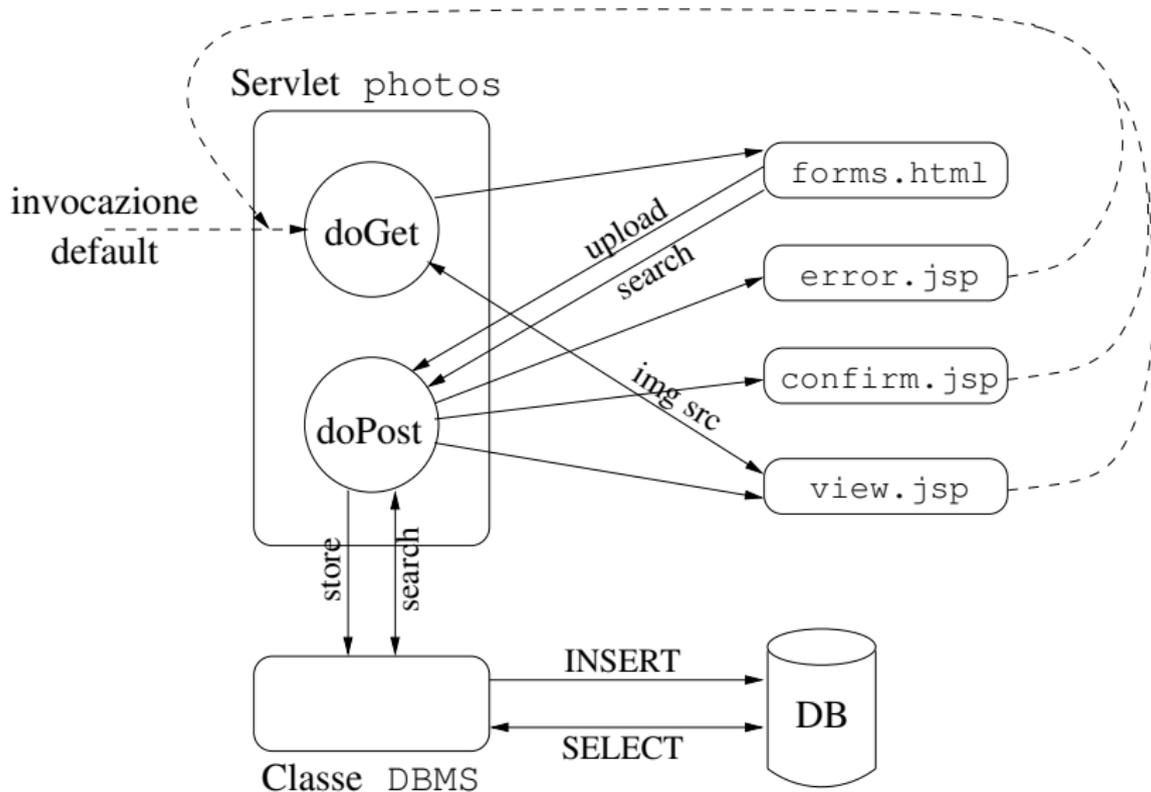
- UNO tra `picturepath` e `picture` DEVE essere NON NULLO
- UNO SOLO tra `picturepath` e `picture` DEVE essere NON NULLO

Path e immagini

Il campo `picturepath` memorizza il path assoluto di un'immagine.

Il campo `picture` memorizza un'immagine vera e propria, a tal fine è definito di tipo `bytea`.

Flusso delle richieste



Metodo doPost

```
public void doPost(HttpServletRequest request,
HttpServletRequest response) {
    ...
    //Essendo i parametri della form di tipo Multipart,
    //ottengo l'oggetto MultipartRequest da cui e'
    //possibile ottenere eventuali parametri
    MultipartRequest multi = new MultipartRequest(request, "/tmp/");
    //ottengo il parametro command che controlla quale
    //azione deve essere eseguita
    String command = (String)multi.getParameter("command");
    ...
}
```

Metodo doPost: SEARCH

```
if (command.equals("SEARCH")) {  
    //sfruttando la classe DBMS eseguo la ricerca con  
    //i parametri dati dall'utente e passo i risultati  
    //a view.jsp per visualizzarli  
    //ottengo gli eventuali ulteriori parametri NAME e SURNAME  
    String name = multi.getParameter("NAME");  
    String surname = multi.getParameter("SURNAME");  
    //tramite la classe DBMS ricerco nel DB  
    //le informazioni richieste  
    Vector result = dbms.search(name,surname);  
    ...  
}
```

Metodo doPost: UPLOAD

```
if (command.equals("UPLOAD")) {
    ...
    //ottengo i valori del checkbox
    String[] store = multi.getParameterValues("storeDB");
    //ottengo il file scelto dall'utente
    File f = multi.getFile("IMAGE");
    ...
    if (f==null) {
        //inoltre un errore da visualizzare alla JSP
    } else {
        //ottengo il tipo del file/immagine inserita nella FORM
        mimetype = (new MimetypeFileTypeMap()).getContentType(f);
        if (mimetype.equals("image/jpeg")) { //se l'immagine e' un JPEG
            fileName = multi.getFilesystemName("IMAGE");
            if (store==null) {
                //costruisco il path assoluto in cui memorizzare
                //l'immagine il metodo System.getenv() permette di
                //recuperare il valore di una variabile d'ambiente.
                //Il file viene memorizzato in una sottocartella "uploads"
                String filepath = System.getenv("CATALINA_BASE") +
                    + "/uploads/" + fileName;
            }
        }
    }
}
```

Metodo doPost: UPLOAD

```
File fOUT = new File(filepath);
//scrivo in fOUT il file f
...
dbms.storePeoplePicture(name, surname, filepath);
//richiamo confirm.jsp per visualizzare la conferma
//dell'upload/inserimento
...
} else { //memorizzo nel DB direttamente l'immagine
dbms.storePeoplePicture(name, surname, f);
//richiamo confirm.jsp per visualizzare la conferma
//dell'upload/inserimento
...
}
}
}
}
```

Metodo doGet: FORMS

Il metodo `doGet` viene richiamato in automatico senza parametri alla prima invocazione della servlet.

In questo caso esso richiama la JSP per mostrare le form di upload/ricerca.

Metodo doGet: img src

La JSP deve mostrare le immagini recuperate dal DB. Per farlo specifica tali immagini in tag HTML `` (vedi slide 33).

Il browser, seguendo il protocollo http, ottengono le immagini richieste dai tag `` in successive automatiche richieste al server inviate tramite il metodo GET.

Dato che le immagini non sono direttamente accessibili dal browser, la servlet, nel metodo `doGet`, deve farsi carico di rispondere anche a queste richieste successive, fornendo le immagini.

Nel nostro caso un'immagine contenuta nel DB viene richiesta fornendo l'`id` della tupla, mentre un'immagine salvata su disco viene richiesta specificandone il `path`.

La risposta della servlet è simile nei due casi, perciò vediamo solo il primo caso.

Metodo doGet: img src

```
if (id!=null) {  
    //ottengo lo stream di output verso la JSP  
    PrintWriter out = response.getWriter();  
    int i; DBMS dbms = new DBMS();  
    //la classe DBMS restituisce un InputStream con cui  
    //costruisco un buffered input stream  
    InputStream is = dbms.searchPicture(Integer.parseInt(id))  
    BufferedInputStream bis = new BufferedInputStream(is);  
    //imposto il tipo della risposta alla JSP  
    response.setContentType("image/jpeg");  
    //imposto la dimensione in byte della risposta alla JSP  
    response.setContentLength(bis.available());  
    //byte per byte copio l'immagine letta dal DB sullo stream  
    //verso la JSP  
    while ((i = bis.read()) != -1)  
        out.write(i);  
    //chiudo lo stream in lettura  
    bis.close();  
}
```

Inserire un file in PostgreSQL

```
void storePeoplePicture(String name, String surname, File f) {
    String insertpic = "INSERT INTO peoplePicture" +
        + "(name,surname,picture) VALUES (?, ?, ?)";
    Connection con = null;
    PreparedStatement pst = null;
    con = DriverManager.getConnection(url, user, passwd);
    pst = con.prepareStatement(insertpic);
    pst.clearParameters();
    pst.setString(1, name);
    pst.setString(2, surname);
    //l'impostazione di campi binari avviene tramite setBinaryStream
    //il secondo parametro e' il FileInputStream da cui PostgreSQL
    //leggera' il file da inserire.
    //Il terzo parametro e' la dimensione in byte del file
    pst.setBinaryStream(3, new FileInputStream(f), (int) f.length());
    //i comandi SQL senza ritorno, come INSERT o UPDATE,
    //devono essere eseguiti con il comando execute()
    //anziche' executeQuery come avviene per le SELECT
    pst.execute();
    con.close();
}
```

Leggere un file da PostgreSQL

```

InputStream searchPicture(int id) {
    String getpic="SELECT picture FROM peoplepicture WHERE id=?";
    PreparedStatement pstmt = null;
    Connection con = null;
    ResultSet rs = null;
    InputStream is = null;
    con = DriverManager.getConnection(urldblab, user, passwd);
    pstmt = con.prepareStatement(getpic);
    pstmt.clearParameters();
    pstmt.setInt(1, id);
    rs = pstmt.executeQuery();
    rs.next();
    //l'immagine, di tipo bytea nel DB, viene ottenuta come
    //un binary stream, in particolare un InputStream
    is = rs.getBinaryStream("picture");
    con.close();
    return is;
}

```

forms.html

```

<FORM NAME="search" ACTION="/photos/servlet/photos"
      METHOD="POST" ENCTYPE="multipart/form-data">
Name: <INPUT TYPE="TEXT" NAME="NAME"><BR>
Surname: <INPUT TYPE="TEXT" NAME="SURNAME"><BR>
<INPUT TYPE=" HIDDEN" NAME="command" VALUE=" SEARCH">
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="SEARCH">
</FORM>

```

```

<FORM NAME="FileUpload" ACTION="/photos/servlet/photos"
      METHOD="POST" ENCTYPE="multipart/form-data">
Name: <INPUT TYPE="TEXT" NAME="NAME"><BR>
Surname: <INPUT TYPE="TEXT" NAME="SURNAME"><BR>
<INPUT TYPE="FILE" NAME="IMAGE" SIZE="35"><BR>
Store directly in DB <INPUT TYPE="CHECKBOX" NAME="storeDB"
                      VALUE="storeDB"><BR>
<INPUT TYPE="HIDDEN" NAME="command" VALUE="UPLOAD">
<INPUT TYPE="SUBMIT" NAME="submit" VALUE="UPLOAD">
</FORM>

```

view.jsp: visualizzazione delle immagini

```

<% Vector result = (Vector)request.getAttribute("data");
PeoplePictureBean ppb = null; %>
<h1>Risultati:</h1>
<table border="1">
<tr><th>Name</th><th>Surname</th><th>Picture</th></tr>
<% for (int i=0; i<result.size(); i++) {
ppb = (PeoplePictureBean)result.get(i);
if (ppb.getPicturePath() == null) { %>
    <tr><td><%=ppb.getName() %></td><td><%=ppb.getSurname() %></td>
    <td align="center">
    
    </td></tr>
<% } else { %>
    <tr><td><%=ppb.getName() %></td>
    <td><%=ppb.getSurname() %></td>
    <td align="center">
    
    </td></tr>
<% } } %>
</table>

```

view.jsp: visualizzazione delle immagini

Il browser quando incontrerà i tag `img` accederà all'URL indicata nell'attributo `src`. Ciò risulterà in una richiesta GET alla servlet (specificando id o path) che risponderà inviando l'immagine da visualizzare.

confirm.jsp: conferma dell'upload

```
<%  
    String msg = (String) request.getAttribute("msg");  
%>  
<h1><%=msg%></h1>
```

Sommario

- 1 Approcci
- 2 Oggetti fondamentali
 - PostgreSQL
 - Form
 - Classi JAVA
- 3 Web Application
 - Nozioni preliminari
 - Servlet
 - DBMS
 - Visualizzazione
- 4 **Installare l'esempio**

Far funzionare la web application

Seguire i seguenti passi:

- 1 nel proprio database `dblabXX` (**non didattica!!!**) creare la tabella `peoplepicture` descritta nella slide 20
- 2 in `tomcat/lib` scaricare (e rinominare) la libreria `cos.jar`
- 3 in fondo al file `.bashrc` nella propria home aggiungere le righe:

```
CLASSPATH=$CLASSPATH:.:$CATALINA_BASE/lib/cos.jar
export CLASSPATH
```

necessarie ad aggiungere la libreria `cos.jar` al classpath

- 4 in `webapps` creare un nuovo context `photos`
In esso ricreare la solita struttura con `WEB-INF`, `classes`, ecc...
- 5 nella cartella `WEB-INF/lib`:
 - creare un link simbolico alla libreria JDBC per PostgreSQL:
`ln -s /usr/share/java/postgresql-jdbc3.jar`
 - scaricare (e rinominare) la libreria `cos.jar`

Far funzionare la web application

- 6 nella cartella `WEB-INF` scaricare e rinominare `web.xml`
- 7 all'interno del context, ma fuori da `WEB-INF`, scaricare e scompattare `vis.tgz` contenente le JSP dell'applicazione
- 8 in `tomcat/src` scaricare e scompattare `photos_src.tgz`.
Si otterrà una cartella `photos` contenente i sorgenti dell'applicazione
- 9 modificare `DBMS.java` inserendo i propri dati (username, password e nome db) per la connessione al proprio DB (non didattica!!!)
- 10 compilare i sorgenti nella cartella `classes` del context `photos`.
- 11 avviare Tomcat
- 12 in Firefox aprire
`http://localhost:8080/photos/servlet/photos`

Far funzionare la web application

- 13 si dovrebbe ottenere questa schermata:

Mozilla Firefox <@(none)>

File Modifica Visualizza Cronologia Segnalibri Strumenti Guida

http://localhost:8080/photos/servlet/photos

Google

Search form

Name:

Surname:

Upload form

Name:

Surname:

Store directly in DB

... speriamo...