Programmazione Web

Lezione del 27 Maggio 2020

Docente: Novella Bartolini

Come condividere dati tra diverse risorse (1)

- Se :
  - la richiesta può essere inoltrata a più pagine di destinazione
  - richiede l'elaborazione di dati contenuti nell'oggetto **`HttpServletRequest`**
- Può essere conveniente spostare l'elaborazione dei dati nella servlet da cui la richiesta ha origine e passare alla pagina di destinazione i dati già elaborati

Come condividere dati tra diverse risorse (2)

- Primo metodo:
  - i dati preventivamente elaborati dalla servlet di origine possono essere inclusi come attributi dell'oggetto **`HttpServletRequest`**
  - **`request.setAttribute("key1", value1);`**
  - La pagina di destinazione può prelevare questi dati
  - **`Type1 value1 = (Type1)request.getAttribute("key1");`**

Come condividere dati tra diverse risorse (3)

- Secondo metodo:
  - rappresentare i dati come bean e memorizzarli nel contesto di visibilità voluto dal tag **`jsp:useBean`** per condividere i bean.

JSP useBean: contesti di visibilità

- request
  - <jsp:useBean id="..." class="..." scope="request" />
- session
  - <jsp:useBean id="..." class="..." scope="session" />
- application
  - <jsp:useBean id="..." class="..." scope="application" />
- page
  - <jsp:useBean id="..." class="..." scope="page" />
    oppure
    <jsp:useBean id="..." class="..." /> (page è il contesto di default)

Come condividere dati tra diverse risorse (4)

- Contesto di visibilità: richiesta
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP che elabori la stessa richiesta.
- Sintassi usata dalla servlet per memorizzare il dato

```
SomeClass value = new SomeClass(…);
request.setAttribute("key", value);
// Usa il RequestDispatcher per inoltrare la richiesta
   alla pagina JSP
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key" class="somepackage.SomeClass"
    scope="request" />
```

Come condividere dati tra diverse risorse (5)

- Contesto di visibilità: sessione
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP all'interno della stessa sessione.
- Sintassi usata dalla servlet per memorizzare il dato

```
SomeClass value = new SomeClass(…);
HttpSession session =
  request.getSession(true);
session.setAttribute("key", value);
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="session" />
```

Come condividere dati tra diverse risorse (6)

- Contesto di visibilità: applicazione
  - Memorizzare un dato attraverso una servlet che sia poi reperibile da una pagina JSP all'interno della stessa applicazione.
- Sintassi usata dalla servlet per memorizzare il dato

```
synchronized(this)* {
  SomeClass value = new SomeClass(…);
  getServletContext().setAttribute("key",
                                   value);
  }
```

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="application" />
```

*la keyword "synchronized" si usa per un segmento di codice che non si vuole venga eseguito simultaneamente da due o più thread. L'oggetto tra parentesi è quello su cui si mette il lock

Come condividere dati tra diverse risorse (7)

- Sintassi usata dalla pagina JSP per recuperare il dato

```
<jsp:useBean id="key"
    class="somepackage.SomeClass"
    scope="application" />
```

- Alternativamente la pagina di destinazione può utilizzare un elemento di scripting
  - `Type1 value1 = (Type1)application.getAttribute("key1");`

Attenzione all'uso di URL relativi in una pagina JSP se questa è la destinazione di una redirezione

- L'inoltro di una richiesta attraverso il dispatcher è trasparente al client (lo stesso vale quando si usa l'azione standard "forward).
- Gli url relativi della pagina JSP vengono riferiti al percorso della servlet da cui la richiesta è stata inoltrata e non al percorso della pagina JSP di destinazione

URL relativi nelle pagine JSP

- Se nella pagina jsp cui la servlet invia la richiesta, è presente un percorso relativo:

```
<IMG SRC="foo.gif" …>
<LINK REL=STYLESHEET
                HREF="JSP-Styles.css"
                TYPE="text/css">
<A HREF="bar.jsp">…</A>
```

  - Il browser lo interpreterà come relativo all'URL della servlet di provenienza della richiesta
- Soluzione:
  - Definire il percorso relativo al server nelle pagine JSP (percorsi che iniziano con "/<contextroot>/...")
- Fare esercizio di prova

Upload di file da una pagina JSP

- Si vuole scrivere una pagina JSP che permetta all'utente di selezionare un file dal proprio disco e inviarlo al server

Upload di file e invio di messaggi di posta

Jakarta Commons

- Sotto-progetto di Jakarta per la creazione e la manutenzione di package che realizzano funzioni genericamente utili e non collegate ad uno specifico framework o prodotto
- I package del repository di Jakarta Commons sono scritti con il proposito di renderli **riutilizzabili** e di metterli **in comune**

fileupload-1.4.jar (Jakarta Commons)

- Si tratta di un package che fornisce componenti pronte per la realizzazione di upload di file dal client verso il server
- Richiede la presenza del file `fileupload-1.4.jar` nella directory /**WEB-INF/lib** della web application
  - In realtà questo file può risiedere anche nelle directory /shared/lib oppure /common/lib di Tomcat, ma è necessario essere l'amministratore del web server (es: non è sempre possibile se si utilizzano servizi di hosting)
  - Documentazione disponibile:

    http://jakarta.apache.org/commons/fileupload/

fileupload-1.4.jar e RFC 1867

- Fornisce capacità di upload a servlet e web application.
- Le classi del package fileupload permettono la gestione di richieste HTTP conformi al documento RFC 1867 , **"Form-based File Upload in HTML"**.
- Una richiesta HTTP di upload è conforme alla RFC1867 se
  1. Viene inoltrata usando il metodo POST,
  2. Specifica un content type "multipart/form-data"

1. Pagina HTML: Creazione del form di richiesta (RFC1867)

- Una richiesta HTTP di upload di file, secondo la RFC1867, deve essere inoltrata con il metodo **`POST`** **`e`** codificata come **`multipart/form-data`**
  - enctype="multipart/form-data“

```
6 <form name="myform" method="post" action="fileuploaddemo.jsp" enctype="multipart/form-data">
7 Specifica il tuo nome: <br>
8
9 <input type="text" name="name" size="15"><br>
10
11 Specifica un file di immagine: <br>
12 <input type="file" name="myimage" ACCEPT="image/gif"><br>
13
14 Specifica un file di testo: <br>
15 <input type="file" name="myfile" ACCEPT="text/html"><br><br>
16
17 <input type="submit" name="submit" value="sottoponi il tuo file">
18 </form>
```

Form per l'inserimento di un nome, un'immagine e un documento di testo

2. Creazione della pagina JSP: Controllare il formato della richiesta

- Importare i package richiesti
- Una richiesta di upload contiene una lista ordinata di *items* che sono codificati secondo la RFC 1867
- Il package Commons fileupload fornisce un metodo per verificarlo:

```
<%@ page import="org.apache.commons.fileupload.*,java.util.*,java.io.*" %>
. . . . . .
 <%
 // check that we have a file upload request
 boolean isMultipart = FileUpload.isMultipartContent(request);
```

3. Analizzare la richiesta

- Creazione di un handler per la richiesta di upload:

```
DiskFileUpload upload= new DiskFileUpload();
```

- Analisi degli item della lista

```
List items=upload.parseRequest(request);
```

4. Analisi degli elementi della lista associata alla richiesta

- Si utilizza un oggetto `Iterator` per analizzare gli item uno ad uno e valutare se si tratti di file oppure di field del form (con isFormField())
  - `Tutti gli item della lista items devono implementare l'interfaccia FileItem anche se non rappresentano dei file`

```
Iterator itr = items.iterator();
while (itr.hasNext()) {
  FileItem item = (FileItem) itr.next();
//controlla se si tratta di un campo del form oppure di un file caricato
  if (item.isFormField()) {
      //prende il nome del campo
      String fieldName=item.getFieldName();
      //se è il campo nome lo si può usare per un messaggio di benvenuto all'utente
      if (fieldName.equals("name")){
        %>
<h1>
 Ciao <%= item.getString() %>
</h1>
<%
      }
  } else  //. . . Siamo nel caso in cui si tratta proprio di un file da caricare sul server
```

5. Scrittura dei file sul server

- Richiedere esplicitamente la scrittura del file su disco, con il percorso voluto

```
} else  //. . . Siamo nel caso in cui si tratta proprio di un file da caricare sul server
{
      //Nota: item.getName() che riporta il full path
      //del file nella macchina del client come nome del file caricato.
      //per superare questo inconveniente utilizziamo fullFile.getName().

      File fullFile = new File(item.getName());
      File savedFile=new File(getServletContext().getRealPath("/"),"definitivi\\"+fullFile.getName());
      item.write(savedFile);
      }
  }
```

6. Per maggiori controlli…

- Si può porre un limite alla dimensione dei file in upload attraverso la funzione upload.setSizeMax
  - Se si imposta tale valore a -1 viene consentito l'upload di file di qualunque dimensione

```
upload.setSizeMax(500000);
```

- Per controllare la dim. massima dei file da caricare in memoria si può imporre un limite massimo oltre il quale i file vengono memorizzati sul disco in una dir temporanea

```
upload.setSizeThreshold(5000);
upload.setRepositoryPath(getServletContext().getRealPath("/")+"tempo");
```

7. Mettere insieme tutti gli elementi

```
1 <html>
2    <head>
3      <title> File html di upload </title>
4   </head>
5 <body>
6 <form name="myform" method="post" action="fileuploaddemo.jsp" enctype="multipart/form-data">
7 Specifica il tuo nome: <br>
8
9 <input type="text" name="name" size="15"><br>
10
11 Specifica l'immagine: <br>
12 <input type="file" name="myimage" ACCEPT="image/gif"><br>
13
14 Specifica il tuo file: <br>
15 <input type="file" name="myfile" ACCEPT="text/html"><br><br>
16
17 <input type="submit" name="submit" value="sottoponi il tuo file">
18
19 </form>
20 </body>
```

`Fileupload.html`

Scrive un form per l'upload di due file e la scrittura di un nome.

```
 1 <html>
 2    <head>
 3      <title> JSP per l'upload dei file del client </title>
 4   </head>
 5 <%@ page language="java" %>
 6 <%@ page import="org.apache.commons.fileupload.*,java.util.*,java.io.*" %>
 7 <body>
 8
 9 <%
10
11 boolean isMultipart = FileUpload.isMultipartContent(request);
12
13
14 DiskFileUpload upload= new DiskFileUpload();
15
16 upload.setSizeThreshold(5000);
17 upload.setRepositoryPath(getServletContext().getRealPath("/")+"tempo");
18 upload.setSizeMax(500000);
19
20
21
22 List items=upload.parseRequest(request);
23
24
25 Iterator itr = items.iterator();
26 while (itr.hasNext()) {
27    FileItem item = (FileItem) itr.next();
28    if (item.isFormField()) {
29         String fieldName=item.getFieldName();
30         if (fieldName.equals("name")){
31           %>
32 <h1>
33  Ciao <%= item.getString() %>
34 </h1>
```

`Fileuploaddemo.jsp`

Esegue l'upload di due file e scrive diversi messaggi all'utente.

```
35
36 <%
37          }
38
39        } else
40        {
41        File fullFile = new File(item.getName());
42 %>
43
44 <br>
45 Il file di cui hai richiesto l'upload è: <%= item.getName() %> .
46 <br><br>
47 Il nome del file senza percorso sul client è il seguente:
48 <%= fullFile.getName() %>
49 <br><br>
50 Lo metteremo invece in questa directory: <%= getServletContext().getRealPath("/")+"definitivi" %>
51 <br>
52
53 <%
54 File savedFile = new File(getServletContext().getRealPath("/"),"definitivi\\"+fullFile.getName());
55        item.write(savedFile);
56        }
57    }
58 %>
59 </body>
```

`Fileuploaddemo.jsp`

Esegue l'upload di due file e scrive diversi messaggi all'utente. (cont.)

Invio di una mail da una pagina JSP

- Si vuole scrivere una pagina JSP che permetta all'utente di inviare una mail attraverso il server

Mandare una e-mail da una pagina JSP

- Esistono numerosi package che fanno al caso nostro
  - Le API JavaMail sono fornite da Sun e sono open source
  - Non fanno parte del JDK
  - Per funzionare richiedono il JAF (JavaBeans Activation Framework)
  - Trovate il file javamail-1_6_2.zip contenente i package (jar) che vi servono sul sito del corso

Come installare i package di JavaMail

- Bisogna aggiungere i file mail.jar e activation.jar in una delle directory:
  - ...web-apps/<context-root>/WEB-INF/lib/
    per rendere visibili le classi alla vostra applicazione
  - <CATALINA_HOME>/shared/lib oppure
    <CATALINA_HOME>/common/lib
    per rendere visibili le classi a tutte le applicazioni web

Per testare l'installazione

- Scrivere una semplice direttiva di importazione in una pagina JSP

```
<html>
<head><title>Test di installazione</title></head>
<body>
<%@ page import="javax.mail.*" %>
Se leggi questo messaggio l'installazione di JavaMail <br>
è stata eseguita correttamente <br>
</body>
</html>
```

1. Definire una sessione di e-mail

- La classe **`javax.mail.Session`** rappresenta una sessione di mail
- Per la creazione di un oggetto della classe **`Session`** è necessario definire un oggetto **`Properties`** di configurazione
- Per l'esempio che segue basta definire una sola proprietà **`"mail.smtp.host"`** che definisce un server di posta SMTP

1. Definire una sessione di e-mail

- Definiamo una sessione di e-mail indicando il nome di un server smtp

```
Properties props = new Properties();
props.put("mail.smtp.host", "mail.fastwebnet.it");
//....altre proprietà se necessario
Session sendMailSession;
sendMailSession=Session.getInstance(props);
```

2. Definizione del messaggio di e-mail

- Classe **`Message`**: si tratta di una classe astratta, che rappresenta tutto ciò che ha a che vedere con il messaggio di e-mail
- JavaMail fornisce una sottoclasse **`MimeMessage`** per rappresentare e-mail MIME
- Per definire un nuovo oggetto **`MimeMessage`** si fa riferimento alla sessione di email corrente:

```
Message newMessage = new MimeMessage(sendMailSession);
```

2. Definizione del messaggio di e-mail

- Una volta creato l'oggetto **MimeMessage**, definire i valori dei suoi attributi (consultare la documentazione per un elenco completo)
- "**Subject**": una stringa

```
newMessage.setSubject("Subject della mail.");
```

- "**Text**": una stringa

```
newMessage.setText("Ciao questa mail proviene da una pagina JSP");
```

- "**From**": oggetto di classe **InternetAddress**

```
InternetAddress from = new InternetAddress("pluto@paperopoli.net");
newMessage.setFrom(from);
```

2. Definizione del messaggio di e-mail

- Per definire il destinatario oltre all'indirizzo serve un parametro ulteriore:
  - RecipientType.TO
  - RecipientType.CC
  - RecipientType.BCC

```
InternetAddress to = new InternetAddress("minnie@paperopoli.com");
newMessage.addRecipient(RecipientType.TO, to);
```

3. Specificare il protocollo da usare

- La classe astratta **`Transport`** serve a definire il protocollo da usare per spedire la mail (fornisce tutti i metodi per utilizzare il protocollo smtp)
- JavaMail ci risparmia l'implementazione dell'intero protocollo SMTP!
- Notare che non è necessario/possibile creare un'istanza di questa classe per spedire una mail. La classe è astratta e il metodo send() ad essa associato è un metodo "static".

Esempio di pagina JSP che spedisce una mail

```
<html><head></head>
<body>
<%@ page import="java.util.*, javax.mail.*, javax.mail.internet.*" %>
<% Properties props = new Properties ();
   props.put("mail.smtp.host", "smtp.fastwebnet.it");
   Session s = Session.getInstance(props, null);
   MimeMessage message= new MimeMessage(s);
   InternetAddress from=new InternetAddress("me@esempio.com");
   message.setFrom(from);
   InternetAddress to=new InternetAddress("you@esempio.com");
   message.addRecipient(Message.RecipientType.TO,to);
   message.setSubject("Test di posta da JavaMail");
   message.setText("Ciao questo messaggio viene da una pagina JSP");
   Transport.send(message);
%>
E' stato spedito un messaggio di email
</body></html>
```

Esempio: creare una pagina JSP per scrivere mail attraverso un form: form.html

```
<html><head><title>Form per spedire una mail</title></head>
<body>
<p><b> Una pagina per spedire le mail </b> </p>
<form action="sendmail.jsp" method="post">
   <table align="center">
      <tr><td>To: </td><td><input name="to" size="50" /></td></tr>
      <tr><td>Subject: </td><td><input name="subject" size="50"
                                 value="Mail scritta dal form" /></td></tr>
      <tr><td colspan="2" align="center">
           <textarea name="text" cols="50" rows="20">
           Ciao da JavaMail!
           </textarea>
      </td></tr>
      <tr><td colspan="2" align="center">
          <input type="submit" value="Send E-Mail"/>
      </td></tr>
   </table>
</body></html>
```

Esempio: creare una pagina JSP per scrivere mail attraverso un form: sendmail.jsp

```
<html><head></head>
<body>
<%@ page import="java.util.*, javax.mail.*, javax.mail.internet.*" %>
<% Properties props = new Properties ();
   props.put("mail.smtp.host", "smtp.fastwebnet.it");
   Session s = Session.getInstance(props, null);
   MimeMessage message= new MimeMessage(s);
   InternetAddress from=new InternetAddress("me@esempio.com");
   message.setFrom(from);
   String toAddress = request.getParameter("to");
   InternetAddress to = new InternetAddress(toAddress);
   message.addRecipient(Message.RecipientType.TO,to);
   String subject=request.getParameter("subject");
   message.setSubject(subject);
   String text= request.getParameter("text");
   message.setText(text);
   Transport.send(message); %>
E' stato spedito un messaggio di email, clicca
<a href="form.html" > qui </a> per mandarne un altro.
</body></html>
```

Come specificare destinatari multipli

- Si possono ripetere diverse chiamate al metodo **`message.addRecipient()`**
- Si possono usare i metodi
  - setRecipients(Message.RecipientType type, Address[] addresses)
  - addRecipients(Message.RecipientType type, Address[] addresses)

Come specificare destinatari multipli (cont.)

- I metodi
  - setRecipients(Message.RecipientType type,
    String addresses)
  - addRecipients(Message.RecipientType type,
    String addresses)

  prendono come secondo parametro una stringa corrispondente ad una lista di destinatari separati da una virgola.