Manual del laboratori de "Programació Concurrent i Distribuïda" JAVA Concurrent

Xavier Messeguer

May 7, 2008

Sessió

WEB

Observeu que en els capítols precedents, l'execució dels programes es feia des de la línia de comandes, tant del programa servidor com del dels clients. En aquest capítol es mostra com es pot implementar l'execució dels clients via web.

El procés que heu seguit fins ara és el següent:

1. Inicieu el programa servidor que atén les peticions en un port determinat del *host* servidor.

2. Inicieu el programa client que es connecta al *host*:port del programa servidor.

3. El programa client presenta un *entorn de comunicacio client/servidor* (en línia de comandes o gràfic) que recull localment les peticions del client i les envia al programa servidor.

4. El programa servidor tracta aquestes dades i les retorna a l'entorn local del client.

El que mostrarem ara és com fer que el *browser* web faci de "*entorn de comunicacio client/servidor*". I per aconseguir-ho seguirem els passos següents:

1. Iniciar el programa servidor però no en qualsevol *host* sino que, per motius de seguretat, en un un port del *host* que suporta el *browser*.

2. Crear una pàgina web de benvinguda, que expliqui què fa el programa i que recolli les dades inicials del client.

3. Dissenyar un programa que llegeixi les dades introduides via web, les tracti i faci la petició de carregar l'*applet*. Aquests programes s'anomenen *cgi*'s i un *applet* és un programa que el client demana al servidor per executar-lo localment.

4. Dissenyar un *applet* que presenti l'"*entorn de comunicacio client/servidor*.

Com es pot veure, en aquest capítol s'integren diferents elements de programació, cadascún dels quals ompliria tot un curs. Per aquesta raó només s'explicaran els aspectes estrictament necessaris per desenvolupar i entendre aquest capítol.

1.1 Browser

Per instal·lar el *browser* d'apache cal que feu `instapache` i per cridar-lo es pot fer:

`http://hostname.fib.upc.es:8080`

bé

`http://127.0.0.1:8080`

Per preparar l'entorn de treball d'aquesta sessió, cal crear els següents subdirectoris:

`apache/htdocs/autoxat/applets`

`apache/cgi-bin/autoxat`

1.2 Servidor

Repecte al programa servidor hi ha pocs canvis, només cal tenir en compte que ha de ser instal·lat en el mateix *host* en que es troba el *browser*. La rao rau en les limitacions que els *browsers* imposen als *applets*, i si no es volen demanar permisos especials només podem connectar-nos d'aquesta manera.

El programa servidor, anomenat `Servidor.java` crea per a cada client un fil servidor amb dos fluxes de comunicació pel mateix canal, un per les entrades i un per les sortides, com es mostra a l'algorisme següent:

Algorisme "ThreadServidor

```
public void run(){
   try {
      InputStream skin = sk.getInputStream();
      DataInputStream dis = new DataInputStream(skin);
      OuputStream skout = sk.getOutputStream();
      DataOutputStream dos = new DataOututStream(skout);
         ⋮
      dis.close();
      skin.close();
      dos.close();
      skout.close();
      sk.close();
   } catch (IOException e)
}
```

Tingueu en compte que serà l'applet qui farà de client i qui, per tant, es comunicarà amb el servidor.

1.3 Pàgina web

Aquí proposem una pàgina web de benvinguda que presenta un botó que el ser premut força l'execució d'un altre programa anomenat `autoxat.cgi`.

Les pàgines web s'escriuen en llenguatge *HTML* i són interpretades pel *browser*. Tot seguit introduirem les instruccions necessaries per crear una pàgina web.

L'esquema bàsic reflecteix que hi ha una capçalera i un bloc d'instruccions:

```
<html>
<HEAD>
<TITLE> Auto-xat </TITLE>
</HEAD>
<body>
    ⋮
</body>
</html>
```

essent un possible bloc d'instruccions el següent:

Algorisme "autoxat.html"

```
<body>
<p>Es un exemple de p&agrave;gina web.
</p>
<br>
Per iniciar prem el bot&oacute;
<FORM METHOD="GET" ACTION="http://127.0.0.1:8080/cgi-bin/autoxat/autoxat.cgi">
<input type="submit" value="Som-hi!">
</BODY>
```

Expliquem les etiquetes:

- L'etiqueta i delimiten un paràgraf i afegeixen una línia en blanc abans i després del paràgraf.
- L'etiqueta
 fa un salt de línia.
- L'etiqueta *FORM* ens permet presentar un botó que inicií l'execució del programa *autoxat.cgi*. En el nostre cas aquest programa és l'executable d'un programa en `C`, però cal tenir en compte que hi ha altres llenguatges, com Perl o PHP, que poden ser més adients si s'han de llegir dades del client, cosa que no hem de fer en el exemple que desenvolupem.

1.4 CGI's

El nom de *cgi* provè de *Common Gateway Interface* i és un programa que s'executa per ordre de la pàgina web, per tant el podem escriure en un llenguatge directament interpretable com `Perl` o `php`, o en un llenguatge compilable com `C` que serà el nostre cas. La característica principal dels cgi's és que reben les dades per l'entrada estandard i retornen els resultats per la sortida estandard que alhora és l'entrada del *browser*.

Veguem un exemple de *cgi* que no rep dades però que retorna una pàgina en llenguatge html per la sortida estandard i que al entrar en el *browser* serà interpretada. Observeu que el fitxer `.cgi` és la versió compilada del `.c`:

Algorisme "autoxat.c"

```
#include <stdio.h>
main() {
   // Creacio de la pagina html indicant que es de tipus html
   printf("Content-type:  text/html\n\n");
   // Codi html
   printf("<HTML>\n");
   printf("<HEAD>\n");
   printf("</HEAD>\n");
   printf("<body>\n");
   printf("<applet code=\"autoxat.class\"\n");
   printf(" codebase=\"http://127.0.0.1:8080/autoxat/applets\"\n");
   printf(" width=800 height=500>\n");
   printf("</applet>\n");
   printf("</BODY>\n");
   printf("</HTML>\n");
}
```

Es clar que el *cgi* està enviant la següent pàgina web que demana un *applet* que es troba en la URL especificada per executar-lo dins una finestra de les mides indicades per *width* i *height*.

```
Content-type:  text/html
<HTML>
<HEADER>
</HEADER>
<body>
<applet code="autoxat.class"
   codebase="http://127.0.0.1:8080/autoxat/applets"
   width=800 height=500>
</applet>
</BODY>
</HTML>
```

1.5 APPLETS

El programa `.cgi` força al *browser* a cridar un *applet*, que no és més que una aplicaciøque s'executa localment en el *browser*. Per motius de protecció els *applets* no poden accedir al disc ni poden fer connexions a d'altres ordinadors excepte al *host* del servidor web.

La classe que implementa un *applet* és una extensió de la classe `Applet`:

```
import java.applet.*;
public class autoxat extends Applet{
   ⋮
   public void init() {
      ⋮
   }
   public void start(){
      ⋮
   }
   public void stop(){
      ⋮
   }
   public void destroy(){
      ⋮
   }
}
```

Hi ha quatre mètodes que poden ser implementats:

- `init()`: s'executa quan es carrega l'*applet*.
- `start()`: s'executa cada cop que s'entra a la pàgina que conté l'*applet*.
- `stopt()`: s'executa cada cop que es surt de la pàgina que conté l'*applet*.
- `destroy`: s'executa quan s'elimina l'*applet*.

Seguint amb el nostre exemple el següent *applet*, que presentarà l'entorn de comunicacio del client, fa les següents accions:

1. Crea un canal de comunicació `sk`.
2. Crea dos fluxes de comunicació `dis` i `dos`.
3. Crea dues finestres `frebretext, fenviartext` de text, una per rebre dades i l'altre per enviar-les.
4. Inicia dos fils, `TLector` que rebra les dades del programa servidor i `TEscriptor` que enviarà les dades al servidor.

El codi complet és:

Algorisme "autoxat.java" de l'exemple

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
public class autoxat extends Applet{
   //Declaracio de les finestres
   TextArea frebretext = new TextArea();
   TextArea fenviartext = new TextArea();
   // Declaracio dels fluxes de dades
   Socket sk = null;
   InputStream skin;
   DataInputStream dis;
   OutputStream skout;
   DataOutputStream dos;
   public void init() {
      try{
         sk=new Socket(...);
         skin=sk.getInputStream();
         dis= new DataInputStream(skin);
         skout=sk.getOutputStream();
         dos= new DataOutputStream(skout);
      } catch (IOException e) {}
      //dimensionar i mostrar finestres
      frebretext.setColumns(40);
      fenviartext.setColumns(20);
      add(frebretext);
      add(fenviartext);
      TLector lec = new TLector(dis,frebretext);
      TEscriptor esc = new TEscriptor(dos,fenviartext);
      lec.start();
      esc.start();
   }
   public void stop(){
      try{
         dos.close();
         skout.close();
         dis.close();
         skin.close();
         sk.close();
      } catch (IOException e) {}
   }
}
```

- El codi del fil lector:

```
import java.io.*;
import java.net.*;
import java.awt.*;
```

```
public class TLector extends Thread{
   private DataInputStream dis;
   private String s;
   private TextArea finestra;
   public TLector (DataInputStream d, TextArea fin){
      dis=d;
      finestra=fin;
   }
   public void run(){
      try{
        finestra.append("Lector engegat\n");
        s=dis.readUTF();
        while (s.length()>0){
           finestra.append(s);
           s =dis.readUTF();
        }
      } catch (IOException e) {}
   }
}
```

- El codi del fil escriptor:

```
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
public class TEscriptor extends Thread{
   private DataOutputStream dos;
   private TextArea finestra;
   public TEscriptor (DataOutputStream d,TextArea fin){
       dos=d;
       finestra=fin;
   }
   public void run(){
       ReturnPremut premut= new ReturnPremut(dos,finestra);
       finestra.addKeyListener(premut);
       finestra.append("Escriptor engegat\n");
       try{
          dos.writeUTF("Connexio establerta");
       } catch (IOException e) {}
   }
   class ReturnPremut extends KeyAdapter{
       private DataOutputStream RKdos;
       private String RKs;
       private TextArea RKfinestra;
       //Netscape obliga a redeclarar les variables
       //dos i finestra dins ReturnPremut
       public ReturnPremut(DataOutputStream d,TextArea fin){
          RKdos=d;
          RKfinestra=fin;
       }
       public void keyPressed(KeyEvent tecla){
```

```
            if (tecla.getKeyCode()==KeyEvent.VK_ENTER){
               try{
                  RKs=RKfinestra.getText();
                  RKdos.writeUTF(RKs);
                  RKfinestra.setText("");
               } catch (IOException e) {}
            }
         }
      }
   }
```

En aquest codi cal es mostra com lligar un event a una acció.
Les instruccions

```
ReturnKey premut= new ReturnPremut(dos,finestra);
finestra.addKeyListener(premut);
   ⋮
public void keyPressed(KeyEvent tecla){
   ⋮
}
```

creen un objecte de la classe `ReturnPremut` que captura l'interrupció generada al prèmer una tecla i lligan l'objecte a la finestra. Així quan es premi la tecla `Return` s'executara el codi del mètode `keyPressed`.

1.6 Exemple "autoxat"

En aquesta última secció concretarem totes els aspectes de l'exemple. `HOME` voldrà dir `usuari/apache`.

- En el subdirectori `HOME/htdocs/autoxat` hi poseu totes les classes del servidor "Servidor.java" i "ThreadServidor" amb les versions compilades.
- En el subdirectori `HOME/htdocs/autoxat` hi poseu la pàgina web de benvinguda "autoxat.html".
- En el subdirectori `HOME/cgi-bin/autoxat` hi poseu el "autoxat.c" i la seva versió compilada "autoxat.cgi".
- En el subdirectori `HOME/htdocs/autoxat/applets` hi poseu totes les classes dels *applets*: "autoxat.java", "TLector.java" i "TEscriptor.java" amb les versions compilades.

1.7 Exercici

1. L'exercici pel proper dia és la transformació de l'exemple de l'autoxat en un xat real que permeti la connexió de molts usuaris i que envii els missatges a tothom. Penseu que en el entorn de la nostra assignatura heu de garantir que els missatge arriben a tothom sencers (es a dir no es troben intercalats) i en el mateix ordre.