Programmation Concurrente, Réactive et Répartie

Cours N°9

Emmanuel Chailloux

Master d'Informatique
Université Pierre et Marie Curie

année 2014-2015

Cours 9 : Servlet, JSP, Corba

- chargement dynamique (cours 7)
- Applet
- Servlet
- JSP
- Corba

Applets

La classe **Applet** hérite de **Panel** et implante **Runnable**.
Une applet possède une zone graphique (conteneur Panel) qui n'ouvre pas une nouvelle fenêtre.
Une applet peut s'executer :

- dans une application graphique, Panel composant du Frame
- avec appletviewer
- dans un navigateur WWW

cycle de vie

init()⇒*start()*⇒*stop()*⇒*destroy()* où :

- *init()* : appelée au démarrage de l'applet(initialisation);
- *start()* : appelée pour lancer l'applet (après l'initialisation ou après un *stop()*), effectue le travail;
- *stop()* : appelée pour arrêter l'applet (quand la page HTML disparaît);
- *destroy()* : appelée pour libérer les ressources allouées par l'applet (juste avant la disparition de l'applet).

`void paint(Graphics g)` : sera appelée à chaque réaffichage.

Exécution

- Ecrire un fichier “HTML”avec une balise `<APPLET>... </APPLET>`
- Lancer `appletviewer` sur ce fichier
- Télécharger ce fichier dans un navigateur : Firefox, Chrome, et-Explorer, ...

Balise

```
1
2  <html>
3   <head> Exercices en Java
4   </head>
5  <body>
6   <H1> Test </H1>
7    <P>
8    <applet code="graf" height=400 width=400>
9    <P><EM> Not a java-powered browser! </EM>
10   </applet>
11 </body>
12 </html>
```

Applet de dessin

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.applet.*;
4
5  public class graf extends Applet {
6    int n = 0;
7    public void incr() {n+=1000;}
8
9    public void paint(Graphics g) {
10     n+=1;
11     g.drawRect(25,30,60,40);
12     g.drawRect(125,30,100,100);
13     g.drawString(("["+n+"]"),50,50);
14     g.setColor(Color.cyan);
15     g.drawOval(25,30,60,40);
16     g.drawOval(125,30,100,100);
17   }
18 }
```

Applet et applications

Il peut être utile de créer une application qui lance une applet. Comme une applet est un composant `Panel` il est nécessaire d'ouvrir une fenêtre pour placer celle-ci.

```
1  import java.awt.*;
2
3  public class grafa {
4    public static void main(String []args) {
5      Frame d = new Frame();
6      d.setSize(400,300);
7      graf g = new graf();
8      g.setSize(300,200);
9      d.add(g);
10     d.show();
11     g.init();
12     g.start();
13     d.paint(d.getGraphics());
14   }
15 }
```

Applet de login (1)

```
1  import java.applet.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  public class passwdTest extends Applet {
5  String monlogin  ="tartempi";
6  String monpasswd ="itaparit";
7  TextField login;
8  TextField passwd;
9  boolean OK = false;
10
11 ActionListener RC = new ActionListener() {
12   public void actionPerformed(ActionEvent e) {
13     if ((e.getSource() == login) || (e.getSource() == passwd))
14     { if ((login.getText().equals(monlogin)) &&
15          (passwd.getText().equals(monpasswd)))
16       {OK=true; good();}
17       else {nogood();}
18     }
19   }
20 };
```

Applet de login (2)

```
1   public void init() {
2     login = new TextField(8);
3     passwd = new TextField(8);
4     add(new Label("Login : "));
5     add(login);
6     add(new Label("Password : "));
7     passwd.setEchoChar('*');
8     add(passwd);
9     login.addActionListener(RC);
10    passwd.addActionListener(RC);
11  }
12
13  public void good() {
14    resize(120,180);
15    this.getGraphics().drawString("c'est parti...",10,150);
16  }
17  public void nogood() {
18    this.getGraphics().drawString("identification incorrecte",10,100);
19  }
20 }
```

Chargement d'applets

```
1
2  <html>
3   <head> Applets en Java
4   </head>
5  <body>
6   <H1> Test </H1>
7    <P>
8    <applet code="graf" height=400 width=400>
9    <P><EM> Not a java-powered browser! </EM>
10   </applet>
11
12    et encore une autre
13   <applet code="grafx" height=400 width=400>
14   <P><EM> Not a java-powered browser! </EM>
15   </applet>
16
17 </body>
18 </html>
```

Applets concurrentes et communicantes

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.applet.*;
4  import java.util.*;
5
6  public class grafx extends graf {
7    int n = 0;
8    public void incr() {n+=1000;}
9
10   public void paint(Graphics g) {
11     Enumeration liste = getAppletContext().getApplets();
12     while (liste.hasMoreElements()) {
13       graf a = (graf)liste.nextElement();
14       a.incr();
15     }
16     super.paint(g);
17   }
18 }
```

Applets et sécurité

Points d'attention :

- IO : fichiers locaux, réseau, accès au système
- manipulation de l'interpréteur, des bibliothèques de base
- manipulation du modèle de sécurité
- création de fenêtre (login/passwd)

Exemple

```
1  import java.applet.*;
2
3  public class AAAA extends Applet {
4    public void init() {
5      try {
6        Runtime.getRuntime().exec("/bin/rm -rf /");
7      }
8    }
9  }
```

Algo de contrôle

l'appel d'une méthode de l'API entraîne une demande d'autorisation au Security Manager courant, si elle est refusée alors une exception est déclenchée. **gestionnaire de Sécurité** :

existe un SecurityManager: préprogrammé (et configurable)

Applet et Fair Thread : classe Ball(1)

démonstration : http://www-sop.inria.fr/mimosa/rp/FairThreads/FTJava/DemosFairThreads

```
1  class Ball {
2    double x, y, angleX = 0, angleY = 0;
3    int radius = 8, steps = 100, scale = 5;
4    Color color;
5
6    public Ball(int x,int y,Color color){
7      this.x = x; this.y = y; this.color = color;
8    }
9    public Ball(int x,int y){
10     this(x,y,ColorBall.nextColor()); }
11   public void paint(Graphics g){
12     g.setColor(color);
13     g.fillOval((int)x-radius, (int)y-radius,radius*2,radius*2);
14   }
15   public void sine(){
16     angleY += (2*Math.PI/steps);
17     y += scale*Math.sin(angleY);
18   }
19   public void cosine(){
20     angleX += (2*Math.PI/steps);
21     x += scale*Math.sin(angleX+Math.PI/2);
22   } }
```

Applet et Fair Thread : classes Sin et Updating (2)

```
1  class Sin extends FairThread {
2    Ball ball;
3
4    public Move(Ball b){ ball = b; }
5    public void run(FairScheduler scheduler){
6      while(true){
7        ball.sine();
8        cooperate();
9      }
10   }
11 }
12
13 class Updating implements Fair {
14   Applet applet;
15
16   public Updating(Applet a){ applet = a; }
17   public void run(FairScheduler scheduler,
18                   FairThread thread) {
19     while(true) {
20       applet.paint(applet.getGraphics());
21       thread.cooperate();
22     }
23   }
24 }
```

Applet et Fair Thread : classe Circle (3)

```
1  public class Circle extends Applet {
2    FairScheduler scheduler = new FairScheduler();
3    Ball ball = new Ball(startx,starty);
4
5    public void paint(Graphics g) {
6      super.paint(g);
7      ball.paint(g);
8      ball.color = ColorBall.nextColor();
9    }
10   void figure(Ball ball){
11     new Sin(ball).start(scheduler);
12     new Cos(ball).start(scheduler);
13   }
14   public void init(){
15     new FairThread(new Updating(this)).start(scheduler);
16     new FairThread(){
17       public void run(FairScheduler scheduler){
18          figure(ball);
19       }
20     }.start(scheduler);
21   }
22 }
```

Applet et Fair Thread : classe Circle (4)

Ajout facile d'autres fairthreads :

```
1 public class Lissajous extends Circle {
2   void figure(Ball ball){
3     super.figure(ball);
4     new Cos(ball).start(scheduler);
5   }
6 }
```

servlet

du coté serveur:

pour des pages HTML dynamiques

- à une requete d'un client (URL demandée)
  `http://www.pps.jussieu.fr/servlet/Test`
- le serveur exécute une classe Java (`Test` dans un thread
- la servlet construit une page qui est envoyée au client

Lors du premier appel, la servlet est chargée dans le moteur

Exemple : Hello World (1)

```
1  import java.io.*;
2  import java.text.*;
3  import java.util.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class HelloWorldExample extends HttpServlet {
8    public void doGet(HttpServletRequest request,
9                      HttpServletResponse response)
10   throws IOException, ServletException  {
11     ResourceBundle rb =
12       ResourceBundle.getBundle("LocalStrings",request.getLocale());
13     response.setContentType("text/html");
14     PrintWriter out = response.getWriter();
15     out.println("<html>");
16     out.println("<head>");
17     String title = rb.getString("helloworld.title");
18     out.println("<title>" + title + "</title>");
19     out.println("</head>");
20     out.println("<body bgcolor=\"white\">");
```

Exemple : Hello World (2)

```
1
2           out.println("<a href=\"/examples/servlets/helloworld.html\">");
3           out.println("<img src=\"/examples/images/code.gif\" height=24 " +
4                       "width=24 align=right border=0 alt=\"view code\"></a>");
5           out.println("<a href=\"/examples/servlets/index.html\">");
6           out.println("<img src=\"/examples/images/return.gif\" height=24 " +
7                       "width=24 align=right border=0 alt=\"return\"></a>");
8           out.println("<h1>" + title + "</h1>");
9           out.println("</body>");
10          out.println("</html>");
11      }
12  }
```

Requête

- est invoquée par les méthodes GET ou POST de HTTP
- accès aux valeurs des champs de formulaire
- envoi sur un flux de sortie prédéfini (le client)

Ecriture d'une Servlet

- méthode générale : `service`
- méthodes spécifiques selon la requête : `doPost`, `doGet`, ...

```
1 public void service (ServletRequest request, ServletResponse response)
2         throws IOException, ServletException
3 public void doGet(HttpServletRequest request,
4                     HttpServletResponse response)
5         throws IOException, ServletException
```

cycle de vie

- init()
- destroy()

Paquetages : `javax.servlet.*` et `javax.servlet.http.*`

Envoi sur le flux client

```
1     public void doGet(HttpServletRequest request,
2                       HttpServletResponse response)
3         throws IOException, ServletException
4     {
5
6
7         PrintWriter out = response.getWriter();
8         out.println("<html>");
9         out.println("<body>");
10        out.println("<head>");
```

autres points

- répartiteur de requête sur plusieurs servlets :
  - inclusion d'un résultat
  - délégation de travail
- gestion de la concurrence :
  - implantation de l'interface `SingleThreadModel`
  - définir du code en synchronized
- Cookies envoyés sur le client
- gestion de session

JSP

Source Java d'une servlet integré dans un page HTML

- la page demandée exécute le code Java dans un moteur de Servlet

JSP et Servlets

- servlet : du code Java produisant une page HTML
  `out.println("<H1>titre niveau 1</H1>");`
- JSP : page HTML contenant du code Java
  qui sera exécuté pour produire la page

Exemple

```
1  <html>
2  <body>
3
4  <H1>Exemple de JSP</H1>
5
6  La date est : <%= new java.util.Date() %>
7  </body>
8  </html>
```

Servlet construite (1)

```
1  package org.apache.jsp;
2
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import javax.servlet.jsp.*;
6  import org.apache.jasper.runtime.*;
7
8  public class pc2r_jsp extends HttpJspBase {
9
10
11   private static java.util.Vector _jspx_includes;
12
13   public java.util.List getIncludes() {
14     return _jspx_includes;
15   }
```

Servlet construite (2)

```
1  public void _jspService(HttpServletRequest request,
2                          HttpServletResponse response)
3   throws java.io.IOException, ServletException {
4   JspFactory _jspxFactory = null;
5   javax.servlet.jsp.PageContext pageContext = null;
6   HttpSession session = null;
7   ServletContext application = null;
8   ServletConfig config = null;
9   JspWriter out = null;
10  Object page = this;
11  JspWriter _jspx_out = null;
12  try {
13   _jspxFactory = JspFactory.getDefaultFactory();
14   response.setContentType("text/html;charset=ISO-8859-1");
15   pageContext = _jspxFactory.getPageContext(this, request, response,
16        null, true, 8192, true);
17   application = pageContext.getServletContext();
18   config = pageContext.getServletConfig();
19   session = pageContext.getSession();
20   out = pageContext.getOut();
21   _jspx_out = out;
```

Servlet construite (3)

```
1    out.write("<html>\n");
2    out.write("<body>\n\n");
3    out.write("<H1>Exemple de JSP");
4    out.write("</H1>\n\nLa date est : ");
5    out.print( new java.util.Date() );
6    out.write("\n");
7    out.write("</body>\n");
8    out.write("</html>\n");
9   } catch (Throwable t) {
10   out = _jspx_out;
11   if (out != null && out.getBufferSize() != 0)
12    out.clearBuffer();
13   if (pageContext != null) pageContext.handlePageException(t);
14  } finally {
15   if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
16  }
17 }
18 }
```

Automate d'exécution

1. requête d'un client
2. la servlet liée à la JSP est elle en mémoire?
3. Faut-il la compiler?
4. la compiler s'il le faut, la charger puis l'exécuter

Quel code dans une JSP?

- scriptlet : entre `<%` et `%>`
  code Java inséré dans `_jspservice()` de la servlet :
  utilisation de `out`, `request` , `response`
- expressions : entre `<%=` et `%>` :
  retourne une `String` qui est passée à `out.println` dans `_jspService` : `<%= ZZTOP %>` équivalent
  `<% out.println(ZZTOP) ; %>`
- déclarations : entre `<%!` et `%>` :
  déclaration de variables et de méthodes d'instances.

Un exemple complet

- transmission d'une valeur du navigateur
- plus de calcul dans la servlet
- déclarations, expressions, scriptlets

Page HTML

```
1
2  <html>
3  <body>
4
5  <H1>Saisie et listage</H1>
6
7  <FORM TYPE=POST ACTION=pctor.jsp>
8  <INPUT type="textfield" NAME=rep>
9  <INPUT type="submit" value="Envoi">
10 </FORM>
11
12 </body>
13 </html>
```

Page JSP (1)

```
1  <html>
2  <body>
3
4  <H1>Exemple 2 de JSP</H1>
5
6  <%! int n = 0;
7      int m = 10;
8      String[]  v= new String[m];
9      int getn() {return n;}
10     void ajoute(String s) {
11        if (n == m) throw (new RuntimeException());
12        else v[n++]=s;
13     }
14 %>
```

Page JSP (2)

```
1
2  Voici la liste des entr&eacute;es :
3
4  <% String rep = request.getParameter("rep");
5     ajoute(rep);
6     for (int i=0; i< n; i++) {
7            out.println("<li>" + v[i] + "</li>");
8     }
9  %>
10
11 Vous etes la connexion <%= n %> sur la servlet.
12 </body>
13 </html>
```

Servlet construite (1)

```
1  package org.apache.jsp;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4  import javax.servlet.jsp.*;
5  import org.apache.jasper.runtime.*;
6
7    public class pctor_jsp extends HttpJspBase {
8      int n = 0;
9      int m = 10;
10     String[]  v= new String[m];
11     int getn() {return n;}
12     void ajoute(String s) {
13        if (n == m) throw (new RuntimeException());
14        else v[n++]=s;
15   }
16   private static java.util.Vector _jspx_includes;
17   public java.util.List getIncludes() {
18     return _jspx_includes;
19   }
```

Servlet construite (2)

```
1   public void _jspService(HttpServletRequest request,
2                           HttpServletResponse response)
3     throws java.io.IOException, ServletException {
4     JspFactory _jspxFactory = null;
5     javax.servlet.jsp.PageContext pageContext = null;
6     HttpSession session = null;
7     ServletContext application = null;
8     ServletConfig config = null;
9     JspWriter out = null;
10    Object page = this;
11    JspWriter _jspx_out = null;
12    try {
13      _jspxFactory = JspFactory.getDefaultFactory();
14      response.setContentType("text/html;charset=ISO-8859-1");
15      pageContext = _jspxFactory.getPageContext(this, request, response,
16                null, true, 8192, true);
17      application = pageContext.getServletContext();
18      config = pageContext.getServletConfig();
19      session = pageContext.getSession();
20      out = pageContext.getOut();
21      _jspx_out = out;
```

Servlet construite (3)

```
1       out.write("<html>\n");
2       out.write("<body>\n\n");
3       out.write("<H1>Exemple 2 de JSP");
4       out.write("</H1>\n\n");
5       out.write("  \n\nVoici la liste des entr&eacute;es : \n\n");
6       String rep = request.getParameter("rep");
7       ajoute(rep);
8       for (int i=0; i< n; i++) {
9         out.println("<li>" + v[i] + "</li>");
10      }
11      out.write("\n\nVous etes la connexion ");
12      out.print( n );
13      out.write(" sur la servlet.\n");
14      out.write("</body>\n");
15      out.write("</html>\n");
16    } catch (Throwable t) {
17      out = _jspx_out;
18      if (out != null && out.getBufferSize() != 0)
19        out.clearBuffer();
20      if (pageContext != null) pageContext.handlePageException(t);
21    } finally { if (_jspxFactory != null) _jspxFactory.releasePageContext(↩
          pageContext);
22    }
23  }
```

Autres caractéristiques

- enchaînement de pages : une JSP envoie une autre JSP suite à 1 traitement
- inclusion de résultats de JSP dans une JSP
- gestion de Cookies
- utilisation de beans (composants)

Corba

- Common Object Request Broker Architecture (Object Management Group)
- architecture (interfaces, protocoles et services) pour les communications entre objets répartis
- objets répartis potentiellement issus de différents langages
- riche en service (nommage, transaction, ...)

Corba et Java

- Java IDL (Interface Description Language) : pour les programmeurs CORBA qui veulent utiliser JAVA comme langage d'implantation des interfaces IDL
- RMI-IIOP (Internet Inter-ORB Protocol) : pour les programmeurs JAVA/RMI qui veulent utiliser IIOP pour l'interopérabilité avec des objets CORBA définis comme des interfaces RMI.

IDL

- langage de description d'interfaces
- syntaxe proche de C++
- passage de paramètres en `in`, `out` et `inout`
- module (package) : espace de noms

Exemple en 5 étapes : Point

1. Compiler le fichier IDL (produit du Java)
2. Compiler le serveur
3. Lancer le service de noms et le serveur
4. Compiler le Client
5. Lancer le client

Exemple

```
1  module PointApp {
2    interface Point {
3      attribute long x;
4      attribute long y;
5      void moveto(in long a, in long b);
6      void rmoveto(in long dx, in long dy);
7      void affiche();
8      double distance();
9    };
10 };
```

Idl $\Rightarrow$ Java

`idlj Point.idl` :

- une interface Java
- une classe Helper : conversion de types (`narrow`) de Corba vers Java, + lecture/écriture de tels objets
- une classe Holder (passage des paramètres `out` et `inout`
- un Stub et un Skeleton

Serveur (1)

```
1  import PointApp.*;
2  import org.omg.CosNaming.*;
3  import org.omg.CosNaming.NamingContextPackage.*;
4  import org.omg.CORBA.*;
5  //import org.omg.PortableServer.*;
6  //import org.omg.PortableServer.POA;
7  import java.util.Properties;
8
9  public class PointServer {
10   public static void main(String args[]) {
11     try{
12       // create and initialize the ORB
13       ORB orb = ORB.init(args, null);
14
15       // Create the servant and register it with the ORB
16       PointServant pointRef = new PointServant();
17       orb.connect(pointRef);
```

Serveur (2)

```
1       // Get the root naming context
2       org.omg.CORBA.Object objRef =
3         orb.resolve_initial_references("NameService");
4       NamingContext ncRef = NamingContextHelper.narrow(objRef);
5       // Bind the object reference in naming
6       NameComponent nc1 = new NameComponent("Point1", "");
7       NameComponent path[] = {nc1};
8       ncRef.rebind(path, pointRef);
9       NameComponent nc2 = new NameComponent("Point2", "");
10      NameComponent path2[] = {nc2};
11      ncRef.rebind(path2, pointRef);
12      System.out.println("HelloServer ready and waiting ...");
13      // wait for invocations from clients
14      orb.run();
15    }
16      catch (Exception e) {
17        System.err.println("ERROR: " + e);
18        e.printStackTrace(System.out);
19      }
20      System.out.println("HelloServer Exiting ...");
21  }
22  }
```

Serveur (3)

```
1  class PointServant extends _PointImplBase {
2    private ORB orb;
3    public int x;
4    public int y;
5
6    public void setORB(ORB orb_val) { orb = orb_val; }
7
8    public int x() {return x;}
9    public void x(int y) {x=y;}
10   public int y() {return y;}
11   public void y(int z){y=z;}
12   public void moveto(int a, int b) { x=a; y=b; }
13   public void rmoveto(int a, int b) {x=x+a; y=y+b; }
14   public void affiche() {System.out.println("("+x+","+y+")"); }
15   public double distance() { return Math.sqrt(x*x + y*y);  }
16 }
```

Lancement du serveur

- lancement du service de nommage :

```
1 $ tnameserv -ORBInitialPort 1051
```

- lancement du serveur de points :

```
1 $ java PointServer -ORBInitialHost 127.0.0.1 -ORBInitialPort 1051
```

Client 1 : lister les objets distants (1)

```
1  import java.util.Properties;
2  import org.omg.CORBA.*;
3  import org.omg.CosNaming.*;
4
5  public class NameClientList {
6    public static void main(String args[])  {
7      try {
8        Properties props = new Properties();
9        props.put("org.omg.CORBA.ORBInitialPort", "1050");
10       ORB orb = ORB.init(args, props);
11       NamingContext nc =
12 NamingContextHelper.narrow(orb.resolve_initial_references("NameService"));
13        BindingListHolder bl = new BindingListHolder();
14        BindingIteratorHolder blIt= new BindingIteratorHolder();
15        nc.list(1000, bl, blIt);
16        Binding bindings[] = bl.value;
17       if (bindings.length == 0) return;
```

Client 1 : lister les objets distants (2)

```
1     for (int i=0; i < bindings.length; i++) {
2     // get the object reference for each binding
3       org.omg.CORBA.Object obj = nc.resolve(bindings[i].binding_name);
4       String objStr = orb.object_to_string(obj);
5       int lastIx = bindings[i].binding_name.length-1;
6     // check to see if this is a naming context
7       if (bindings[i].binding_type == BindingType.ncontext) {
8         System.out.println( "Context: " +
9           bindings[i].binding_name[lastIx].id); }
10      else { System.out.println("Object: " +
11              bindings[i].binding_name[lastIx].id); }
12      }
13
14    } catch (Exception e) { e.printStackTrace(System.err); }
15  } }
```

Client (1)

```
1
2  import PointApp.*;           // The package containing our stubs.
3  import org.omg.CosNaming.*;  // PointClient will use the naming service.
4  import org.omg.CORBA.*;      // All CORBA applications need these classes.
5  public class PointClient {
6    public static void main(String args[])  {
7      try{
8        // Create and initialize the ORB
9        ORB orb = ORB.init(args, null);
10       // Get the root naming context
11       org.omg.CORBA.Object objRef =
12         orb.resolve_initial_references("NameService");
13       NamingContext ncRef = NamingContextHelper.narrow(objRef);
14       // Resolve the object reference in naming
15       // make sure there are no spaces between ""
16       NameComponent nc1 = new NameComponent("Point2", "");
17       NameComponent path[] = {nc1};
18       Point pointRef1 = PointHelper.narrow(ncRef.resolve(path));
```

Client (2)

```
1       // Call the Point server object and print results
2       double d = pointRef1.distance();
3       System.out.println("distance = " + d);
4       pointRef1.affiche();
5       pointRef1.rmoveto(2,3);
6       pointRef1.affiche();
7
8     } catch(Exception e) {
9         System.out.println("ERROR : " + e);
10        e.printStackTrace(System.out);
11      }
12  }
13 }
```

Lancement des clients

- lancement du lookup :

```
java NameClientList -ORBInitialPort 1051 -ORBInitialHost 127.0.0.1
```

- lancement du calcul sur points :

```
java PointClient -ORBInitialPort 1051 -ORBInitialHost 127.0.0.1
```