計算機プログラミング'99　＃9

久野　靖*

1999.11.10

0　はじめに

長らくお楽しみ頂きました:-)「計算機プログラミング」も今回で最後になりました。皆様からのリクエストを拝見したのですが、統一してこれ!というのはなかったので、思案した結果、「実用ぽいプログラムを作ってみる」というテーマにしました。

1　例題: 酒屋の倉庫問題

ある酒類販売会社の倉庫には、毎日数個のコンテナが搬入されてくる。コンテナには最大 10 銘柄までのビン詰めの酒が昆載されている。また、顧客からの注文が「顧客名、銘柄、数量」の組で入ってくる。倉庫係が扱う次のような端末アプリケーションを作成せよ。

- コンテナ入荷時に、「コンテナ番号、(銘柄, 数量)…」の情報を入力する。
- 注文時に「顧客番号、銘柄、数量」を入力すると、出荷すべき「(コンテナ番号、数量)…」の情報を表示する。
- さらにそのとき、空になって搬出すべきパレットがあれば、その情報も表示する。

2　アプリケーションの基本設計とインタフェース

上記のアプリケーションを前回やったように「アプレット (クライアント) から RMI 経由でデータベースオブジェクト (サーバ) を呼び出す」形で設計する。まず、クライアントとサーバのインタフェースを定める。

```
import java.rmi.*;

public interface R9Service extends Remote {
  public String arrive(String id, String[] brand, int[] qty)
    throws RemoteException;
  public String ship(String cust, String brand, int qty)
    throws RemoteException;
}
```

コンテナが到着したときは arrive()、出荷のときは ship() を呼ぶ。パラメタの意味は見れば分かりますね?

*筑波大学大学院経営システム科学専攻

3　アプレット側のコード

では次にアプレットのコードを示す。

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;

public class R9Sample1 extends Applet {
  R9Service db = null;
  Label cont_label = new Label("Arrival:");
  TextField cont_id = new TextField();
  TextField cont_brand = new TextField();
  TextField cont_quant = new TextField();
  List cont_list = new List();
  Button cont_add = new Button("Add Brand");
  Button cont_del = new Button("Remove Brand");
  Button cont_send = new Button("Send");
  Label ord_label = new Label("Shipment:");
  TextField ord_cust = new TextField();
  TextField ord_brand = new TextField();
  TextField ord_quant = new TextField();
  Button ord_ship = new Button("Ship");
  Label serv_label = new Label("Server not set yet:");
  TextField serv_port = new TextField();
  Button serv_set = new Button("SET");
  TextArea mesg = new TextArea();
```

ここまではGUI部品の宣言。次にinit()の冒頭部分で部品を画面に配置する。

```
  public void init() {
    setLayout(null);
    add(cont_label); cont_label.setBounds(10, 10, 70, 30);
    add(cont_id); cont_id.setBounds(80, 10, 120, 30);
    add(cont_send); cont_send.setBounds(210, 10, 90, 30);
    add(cont_brand); cont_brand.setBounds(10, 50, 120, 30);
    add(cont_quant); cont_quant.setBounds(140, 50, 60, 30);
    add(cont_add); cont_add.setBounds(210, 50, 90, 30);
    add(cont_del); cont_del.setBounds(210, 90, 90, 30);
    add(cont_list); cont_list.setBounds(10, 90, 190, 210);
    add(ord_label); ord_label.setBounds(310, 10, 70, 30);
    add(ord_cust);  ord_cust.setBounds(380, 10, 120, 30);
    add(ord_brand); ord_brand.setBounds(310, 50, 120, 30);
    add(ord_quant); ord_quant.setBounds(440, 50, 60, 30);
    add(ord_ship);  ord_ship.setBounds(310, 90, 190, 30);
    add(mesg); mesg.setBounds(210, 130, 290, 160);
    add(serv_label); serv_label.setBounds(10, 300, 120, 30);
    add(serv_port); serv_port.setBounds(140, 300, 60, 30);
    add(serv_set); serv_set.setBounds(210, 300, 40, 30);
```

では以下で各ボタンに動作をつける。最初はサーバ側との接続処理を行うボタン。今回はポート番号を画面から入力できるようにしてある。

```
serv_set.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      String url = "rmi://smm:" + serv_port.getText() + "/R9DB";
      db = (R9Service)Naming.lookup(url);
      serv_label.setText(url);
    } catch(Exception e) { mesg.setText(e.toString()); }
  }
});
```

次に、コンテナのデータのうち、銘柄と数量を 1 組ぶん入力するボタン。

```
cont_add.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      int qty = (new Integer(cont_quant.getText())).intValue();
      if(cont_brand.getText().equals(""))
        throw new Exception("Brand not speficied.");
      cont_list.add(cont_brand.getText() + " = " + qty);
      cont_brand.setText(""); cont_quant.setText("");
    } catch(Exception e) { mesg.setText(e.toString()); }
  }
});
```

間違って入力したものを削除するボタン。

```
cont_del.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      if(cont_list.getSelectedIndex() < 0)
        throw new Exception("No item to delete.");
      cont_list.remove(cont_list.getSelectedIndex());
    } catch(Exception e) { mesg.setText(e.toString()); }
  }
});
```

コンテナのデータをまとめて入庫処理を呼び出すボタン。一度 List 部品に文字列として入れてしまったデータを配列に入れ直すのがちょっと面倒くさい。

```
cont_send.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      if(cont_id.getText().equals(""))
        throw new Exception("ContainerID not speficied.");
      int len = cont_list.getItemCount();
      String[] b = new String[len];
      int[] q = new int[len];
      for(int i = 0; i < len; ++i) {
```

```
            String s = cont_list.getItem(i);
            int l = s.lastIndexOf(" = ");
            b[i] = s.substring(0, l);
            q[i] = (new Integer(s.substring(l+3))).intValue();
          }
          mesg.setText(db.arrive(cont_id.getText(), b, q));
          cont_id.setText(""); cont_list.removeAll();
        } catch(Exception e) { mesg.setText(e.toString()); }
      }
    });
```

出庫処理は顧客、銘柄、数量を持って ship() を呼ぶだけだから簡単。

```
    ord_ship.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        try {
          if(ord_cust.getText().equals(""))
            throw new Exception("CustomerID not speficied.");
          if(ord_brand.getText().equals(""))
            throw new Exception("Brand Name not speficied.");
          int qty = (new Integer(ord_quant.getText())).intValue();
          mesg.setText(db.ship(ord_cust.getText(), ord_brand.getText(), qty));
          ord_cust.setText(""); ord_brand.setText(""); ord_quant.setText("");
        } catch(Exception e) { mesg.setText(e.toString()); }
      }
    });
  }
}
```

以上で GUI の部分がアプレットとして完成した。画面のようすは図 1 に示しておく。

4 サーバ側の「ほねぐみ」

では次に、サーバ側の「ほねぐみ」を示そう。ほねぐみ、と言っているのは、全部作ってしまったら皆様のやることがないので、インタフェースは取るけれども中身はからっぽのサーバをここに示すということ。中身をちゃんと処理するのはこれから皆様にこれを改造して作っていただく。

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.io.*;

public class R9ServiceRemote
    extends UnicastRemoteObject implements R9Service {
  public R9ServiceRemote() throws RemoteException { }
  public void load(String s) { }
  public void save(String s) { }
  public String arrive(String id, String[] brand, int[] qty) {
    StringWriter buf = new StringWriter();
```

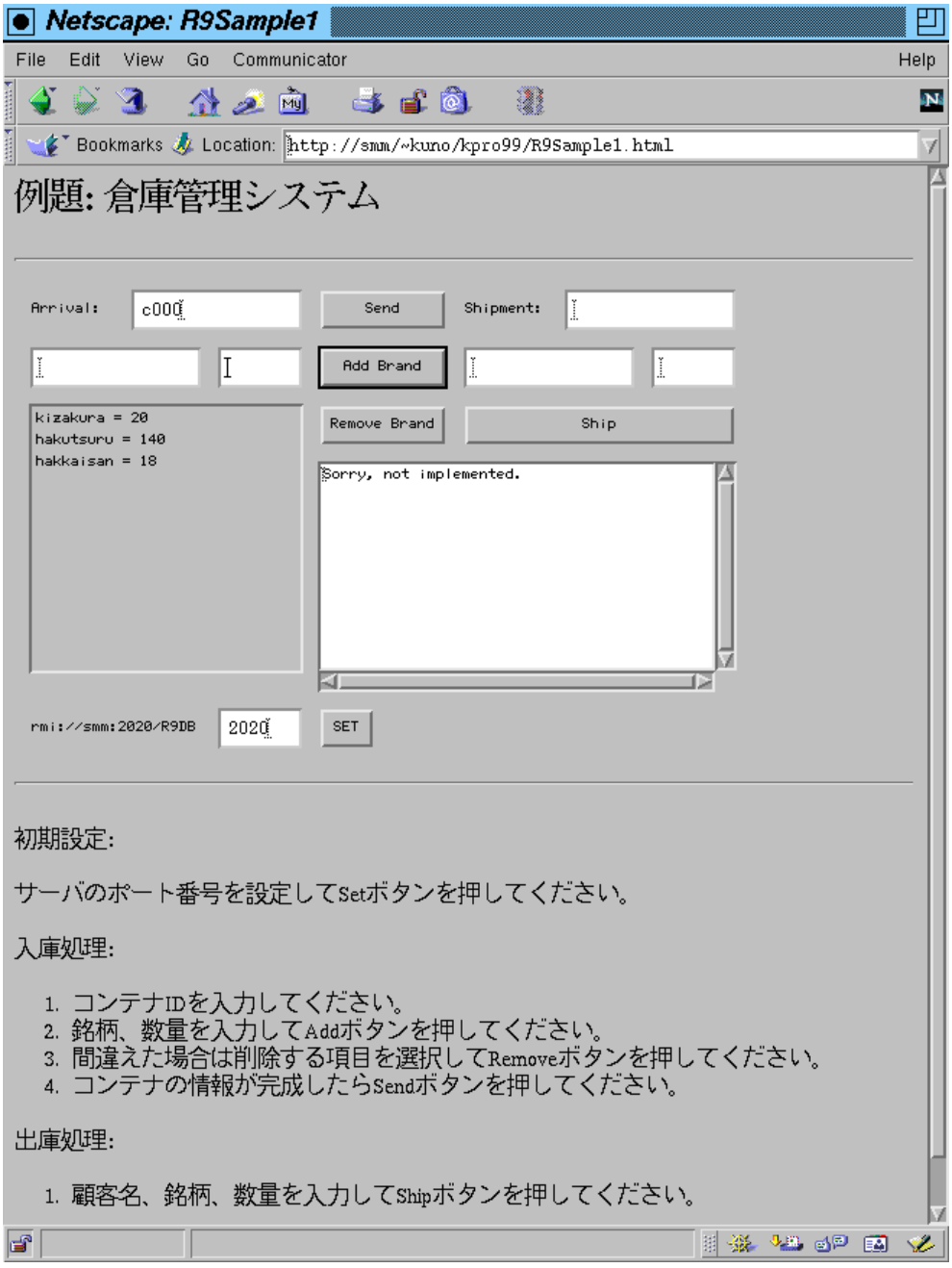

図 1: アプレット画面のようす

```
    PrintWriter out = new PrintWriter(buf);
    out.println("Dummy service echo back.");
    out.println("Container ID = " + id);
    for(int i = 0; i < brand.length; ++i) {
      out.println("  " + i + ">" + brand[i] + " = " + qty[i]);
    }
    out.close(); return buf.getBuffer().toString();
  }
  public String ship(String cust, String brand, int qty) {
    return "Sorry, not implemented.";
  }
  public static void main(String[] args) {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    try {
      R9ServiceRemote db = new R9ServiceRemote();
      db.load("obj.data");
      Registry reg = LocateRegistry.createRegistry(2020);
      reg.bind("R9DB", db);
      System.out.println("Bind Complete");
      while(!in.readLine().equals("quit")) { }
      System.out.println("Saving Data...");
      db.save("obj.data");
      System.exit(0);
    } catch(Exception e) { e.printStackTrace(); }
  }
}
```

これは main() が一緒に入っているので、前回でいうと 2 つぶんのファイルを 1 つのクラスにまとめたことになる。ところで、この中で StringWriter と PrintWriter を使っているが、これは普通の画面出力と同様に println() を使って結果を「書き出して」行き、最後にそれらをまとめて 1 つの文字列として呼び出し側に返すため。

上の例では入庫側でこの方法を使っているが、「本番」クラスでは入庫は「うまく行った」程度のメッセージでよいので、むしろ「出庫」のところで「どのコンテナとどのコンテナから何をいくつ出せ」「どのコンテナは空っぽだから搬出せよ」といったメッセージをまとめて返すために使うことになるだろう。

演習 1 このサーバオブジェクトのファイルをコピーしてきてそのまま動かせ。

演習 2 これを土台にして、実際に処理を行うサーバを作るとしたら、自分ならどのように設計するか? その設計を概説せよ。

演習 3 実際にサーバを構築して動かせ。アプレット側はわざわざコピーしてこなくても久野の WWW ページにあるのを動かせばよい。