Java のＧＵＩプログラミングと JavaBeans 入門

JavaBeans とは、Builder とよばれるＧＵＩツールで(vitsual)プログラミングができるようにするための標準のインタフェースを実装した Java のクラスである。この JavaBeans の仕様にしたがってソフトウエアの部品をつくっておくことによって、これらを適宜組み合わせることによって、プログラミングすることができるようになり、再利用可能なソフトウエア部品を開発する環境を提供する。
JavaBeans に関する情報は、

http://java.sun.com/j2se/1.3/ja/docs/ja/guide/beans/

にある。実際に使う場合には、ここからダウンロードできる。試してみたい人は、

http://java.sun.com/products/javabeans/software/bdk_download.html

から、Beans Development Kit (BDK)をロードする。Linux 等では、platform independent BDK をロードすれば JDK1.2 がインストールしてあればつかうことができる。このパッケージには簡単な Builder である BeanBox が含まれており、JavaBeans による visual プログラミングを試すことができる。これは非常に簡単なものなので、本格的に使いたい場合には市販の Builder がいろいろと発売されている。

JavaBeans では何ができるのか

Beans の場合は次のようなプログラミングが可能となる。

- Builder のデザインシートの上に、お絵描きソフトの感覚でクラスを張り付けることができる。
- 変数値（プロパティ）をプロパティシートに羅列された個々のプロパティエディタ上で変更し、保存できる。
- ２つのクラスに対して特定の「イベントオブジェクト」の交換を指定することで連携させることができる。
- デザインシートごと保存して、実行可能な Java プログラムとすることができる。

JavaBeans は主に、ＧＵＩのプログラミングをする場合に使われるが、必ずしもＧＵＩのプログラミングに限定されるものではない。

JavaBeans 開発には次の２つの側面がある。

- Beans ライブラリを組み合わせて Java プログラムを開発する。 すでに、JavaBeans として開発されたボタンやアイコンなどを組み合わせて簡単にプログラムを作ることができる。
- オリジナルの Beans ライブラリを開発する。必要な部品は自分で開発して、これを組み込んだり、他の人に提供したりすることができる。

なぜ、Java ではこのようなプログラミング環境が可能なのか？これは、Java システムが機械語ではなく、JavaByte コードと呼ばれる仮想機械 JVM(Java Virtual Machine)のコードにコンパイルされ、JVM で実行されるということに大きく依っている。これが非常に柔軟なシステムすることを可能にしている。たとえば、Byte コードにコンパイルされたクラスファイルを Java プログラムから読み、これを解析したり、ロードして動的に実行することができる。コードが自分自身のコードを調べたり、変更したりする機能は Reflection と呼ばれる。また、クラスファイルは動的にロードされるようなシステムになっているため、JVM を持つ web ブラウザが遠くにあるクラスをロードして実行するといった機能も可能になっている。

まず、JavaBeans の主なアプリケーションである Java の GUI プログラミングについて解説する。そのあとで、JavaBeans 技術の元になっている Reflection や Serialization について述べる。

Java のＧＵＩプログラミング：AWT

Java の開発環境である JDK(Java Development Kit)には、ＧＵＩをもつプログラムの作成を容易にするために、Abstract Window Toolkit(AWT)というパッケージが提供されている。AWT は、ボタンやメニューなどのＧＵＩ部品、イメージの表示、マウスやキーボードのからの入力（イベント）を処理する機能を提供している。これらの部品は特定の window システムに依存しないようになっている。最近では、ＡＷＴに変わって Swing というライブラリが使われているようである。
AWT が提供しているクラスは、以下のものである。

- UI 部品：ボタンやメニューなどの部品、それらを配置する入れ物に当たるパネルやウインドウクラス。
- 配置管理：部品をどのように配置するかを管理するための Layout クラス
- その他：グラフィックスライブラリ。

ますは、簡単なプログラムをあげて解説する。このプログラムはボタンを２つもち、ひとつのボタンを

押すと、helloと表示し、もうひとつのボタンを押すとその文字を消すという簡単なものである。
プログラムは、AWT を使うために java.awt.*をインポートする。イベント処理のために、java.awt.event.*をインポートしておく。クラス名は、helloGUI である。
Main では、まず、境界線をもつ Window である Frame を作る。

```
Frame f = new Frame(“hello world”);
```

この Frame は、Container のサブクラスで、Container は他のコンポーネントを配置するクラスである。
次に、自分のクラスのインスタンスを生成し、それを初期化したあと、この Frame の中にいれる。

```
helloGUI hello = new HelloGUI();
hello.init();
hello.start();
f.add(hello,”Center”);
```

なお、Frame にいれたあとで、サイズを設定し、これを表示させると中にはいっているコンポーネントはすべて表示される。

```
f.setSize(300,100);
f.show();
```

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
import java.util,* ;
class HelloGui extends Applet implements ActionListener {
   Button button1,button2;
   Label label;
   public void init(){
      setLayout(new BorderLayout());
      setLocale(Locale.JAPANESE);
      button1 = new Button(“hello”);
      button2 = new Button (“clear”);
      button1.addActionListener(this);
      button2.addActionListener(this);
      label = new Label(“”);
      add(label,”North”);
      add(button1, “Center”);
      add(button2,”East”);
   }
    public void actionPerformed(ActionEvent e){
         if(e.getSource() == button1) label.setText(“hello”);
          if(e.getSource() == button2) label.setText(“”);
     }
     public static void main(String args[]){
         HelloGui h  =  new HelloGui();
         Frame f  =  new Frame("hello GUI");
          hello.init();
          hallo.start();
          f.add(hello,”Center”);
          f.setSize(300,100);
          f.show();
      }
}
```

アプレットと Java アプリケーション

このプログラムは、Applet を extends しており、アプレットとしても使うことができるようになっている。ついでに、アプレットと Java アプリケーションの違いを述べておこう。
アプレットとは、web ページに組み込まれる Java プログラムである。web ページに

```
<APPLET CODE="HelloGUI.calss" WIDTH=300 HEIGHT=100>
</APPLET>
```

と書き、このページと同じところにおいておけば、web ブラウザがこの java プログラムをネットワークを使ってダウンロードし実行してくれる。アプレットにするためには、java.applet.*をインポートし、applet を extends して作成する。アプレットクラスは Continer クラスのサブクラスである Penel クラスをサブクラスとなっており、ブラウザにおいて、このアプレットクラスを生成し、ロードされたアプレット（つまり、HelloGUI）ブラウザのウインドウ（これが Frame にあたる）を表示してくれる。したがって、アプレットとして実行する場合には Frame は必要ない。実行を開始する前に、アプレットは init メソッドを実行し、そのあとで、start メソッドを実行する。init には、通常のコンストラクタに記述する内容、すなわち、インスタンス変数の初期化、コンポーネントの設定、などを行う。通常、アプレットとして実行されるときには、すでにアプレットのインスタンスはできているため、この init で初期化を行う。このほかに、アプレットを一時停止する処理を書く stop メソッドがあり、別の web ページに移ったときに関係するスレッドを停止する処理などを書くことができる。
なお、アプレットとアプリケーションの大きな違いとしてセキュリティがあげられる。任意の他人のブラウザで実行されることを仮定しているために、たとえば、アプレットを実行しているコンピュータでファイルの読み書き、他のプログラムの起動、ロードしたコンピュータ以外への通信などが禁止されている。

コンポーネントのレイアウト

さて、元に戻ることにしよう。初期化 init において、まず、レイアウトのための設定する。

```
setLayout(new BorderLayout());
```

この BorderLayout は、コンテナを５つの領域（上、下、右、左、中央）にわけてそのどこかに配置するレイアウトである。これをコンテナに設定するメソッドが setLayout で、部品を入れるときには、add を使う。入れる部品について、インスタンス変数を宣言しておき、

```
Button button1;  // 部品のインスタンスの変数を宣言
```

次に、init において、インスタンスを生成し、

```
button1 = new Button("hello!!!");  // 部品を作る
```

部品をコンテナにいれる。

```
add(button1,"East");
```

なお、label は、文字を表示するための部品である。

イベント処理

通常のＣの入出力では、getc や read の処理が呼び出さないと入力は行われない。これに対して、Java のＧＵＩ部品は自律的に動いていると考えることができる。すなわち、マウス操作はキーボード入力を行うと何かの処理が実行される。これをイベントとよび、このイベント処理を記述することがＧＵＩプログラミングの中心的な部分となる。（実際、Java では別にスレッドが動いており、それらが入力について監視していると考えることができる）オブジェクト指向言語である Java では、イベント自身もイベント処理もオブジェクト指向のフレームワークで構成されている。
　イベントはイベントが発生したところ（event source）から、そのイベントをうけとって処理をするところ（listerner）に伝えられる。リスナーにはイベント処理のメソッドが定義されており、イベントに応じた処理をすることになる。これは、代理人リスナーモデル(delegation-based Listener Model)と呼ばれ、Java1.1 で導入されたモデルである。イベントを受け取って、処理をするリスナーの機能を持たせるためには java.util.EventListener インタフェースである<EventType>Listener インタフェースをインプリメントする。たとえば、ボタンをクリックした時に生成される ActionEvent を受け取るのは ActionEventListener インタフェースを実装したクラスで、このリスナークラスの actionPerformed メソッドにボタンを押したときに起こる振る舞いを記述する。
プログラムにおいて、helloGUI クラスにボタンをおした時の ActionListener を実装している。イベントソースであるボタンに対し、そのイベントを伝える相手を指定する部分が、

```
button1.addActionListener(this)
```

である。これで、button1 に関するイベントが起きたときに、リスナーである、このクラスのインスタンスに伝えられ actionPerformed が呼び出されるようになる。ここで引数になっているのが伝えられたイベント actionEvent である。この actionEvent にはどこから伝えられたイベントなのかという情報が含まれており、actionPerformed ではこれをつかって、label にある文字を消したり、表示したりしている。

```
if(e.getSource() == botton1) label.setText(“hello”);
```

イベントには、マウスが移動したり、スクロールバーが移動したりといった様々なイベントがあり、リスナーがある。このプログラムでは、リスナーをこのクラスのオブジェクト自身が受け取っているが、別に設定することも可能である。

このような GUI プログラミングのようにオブジェクトに対し、イベントが発生し、それを処理するというモデルは JavaBeans でも基本的なモデルになっている。

Java の Reflection 機能

Reflection とは、「反映」「反射」という意味であるが、プログラムが（他の）プログラムを調べるという意味である。Java の reflection 機能を提供する java.lang.reflect を用いることによって、あるクラス（クラスファイル）にどのようなフィールド（インスタンス変数）、メソッド、Constructor があるかをしらべたり、オブジェクトのフィールドの値を読み書きしたり、メソッドを適当な変数を与えて呼び出すことができる。

この機能を使うためのサンプルプログラムについて解説する。まず、対象とするプログラムは ab.java というファイルに定義された ab というクラスである。ここには a,b という変数、a という変数をセットする関数 setA,読み出す関数 getA、これらを足し算する関数 plus、値をプリントする関数 print が定義されている。

```
% javac ab.java
```

として、コンパイルしておこう。

この内容を調べるのが、report.java である。reflection 機能を使うために、java.lang.reflect.*をインポートしておく。まず、クラスの情報を取得するのが Class.forName である。

```
Class cls;
cls = Class.forName(“ab”);
```

この関数ではクラスがみつからなかったときに exception を返すので、catch しておくようにしておかなくてはならない。forName は Class クラスの静的メソッドとして定義されている。cls にはクラスの情報を表現しているオブジェクトである。フィールド、すなわちインスタンス変数の情報を取得するメソッドが、getDeclaredFields である。これは、Fields オブジェクトの配列を返す。Fields オブジェクトは cls にあるフィールドの情報が入る。これを toSting メソッドで文字列に変換して、出力する。このプログラムを実行すると、

```
%java report
 Field:
   ab.a
   ab.b
```

と表示される。

では、インスタンス変数を読んだり、セットしたりしてみる。このプログラムが test.java である。同じように、Class.forName でクラスの情報を取得しておく。名前を指定して、フィールド情報を取るのが getField である。

```
Field a = cls.getField(“a”);
```

constructor の情報を取得する関数が、getConstructor である。このメソッド関数では引数のタイプ情報を与える。タイプ情報は、Class の配列で、primitive タイプの場合にはデータ型のラップクラス（primitive をクラスオブジェクトとして扱うためのクラス）に定義されている。

```
Class pType1[] = { Integer.TYPE, Integer.TYPE};
Constructor Cons = cls.getConstructor(pType1);
```

メソッド情報を取得する関数が getMethod で、こちらはメソッド名と引数の情報を与える。

```
Method setA = cls.getMethod(“setA”, pType2);
```

上のコンストラクタを呼び出す場合には、Constructor クラスの newInstance メソッドを呼び出す。引数については、すべて、オブジェクトの配列として与える。したがって、primitive タイプの場合は、ラップクラスを使う。

```
Object args[] = { new Integer(10), new Integer(20) };
```

```
object obj = Cons.newInstance(args);
```
返されるのも、オブジェクトである。フィールド情報を使って、フィールドを読み出す場合には、getInt をつかう。ここでは、フィールドを「int として」読み出すことと注意。
```
a.getInt(obj);
```
メソッドを呼び出すときには、invoke メソッドを使う。
```
setA.invoke(obj,arg);
```
なお、plus メソッドのように値を返すメソッドの場合には、帰り値は Object タイプになっており、適当なタイプにキャストして、つかわなくてはならない。

```
---ab.java ------------------------------------------------
public class ab {
   public int a;
   public int b;
   public ab(int x, int y){
      a = x;
      b = y;
   }
   public ab() { a = 1; b = 1; }
   public int getA() { return a; }
   public void setA(int x) { a = x; }
   public int plus() { return a + b; }
   public void print() {
      System.out.println("a="+a+",b="+b);
   }
}


---report.java----------------------------------------------
import java.lang.reflect.*;

public class report {
   public static void main(String argv[]){
      Class cls;
      try {
         cls = Class.forName("ab");
      } catch(Exception e){
         System.out.println("cannot instantiate class");
         return;
      }
      try {
         System.out.println("Field:");
         Field fields[] = cls.getDeclaredFields();
         for(int i = 0; i < fields.length; i++){
              System.out.println(" "+fields[i].toString());
         }
      } catch(Exception e){
         System.out.println("exception!!!");
         return;
      }
   }
}


---- test.java -----------------------------------------
```

```
---- test.java -----------------------------------------
import java.lang.reflect.*;

public class test {
   public static void main(String argv[]){
      Class cls;
      Object obj;
      try {
         cls = Class.forName("ab");
      } catch(Exception e){
         System.out.println("cannot instantiate
class:"+e);
         return;
      }
      try {
         Field a = cls.getField("a");
         Field b = cls.getField("b");
         Class pTypes1[] =
                  { Integer.TYPE, Integer.TYPE };
         Constructor Cons =
                   cls.getConstructor(pTypes1);
         Class pTypes2[] = { Integer.TYPE };
         Method setA = cls.getMethod("setA",pTypes2);
         Class pTypes3[] = { };
         Method getA = cls.getMethod("getA",pTypes3);
         Method plus = cls.getMethod("plus",pTypes3);
         Method print = cls.getMethod("print",pTypes3);
         Object VOID[] = { };
         Object args[] = { new Integer(20), new
Integer(1) };
         obj = Cons.newInstance(args);
         System.out.println("a="+a.getInt(obj));
         System.out.println("b="+b.getInt(obj));
         a.setInt(obj,10);
         print.invoke(obj,VOID);
         System.out.println("a="+a.getInt(obj));
         System.out.println("b="+b.getInt(obj));
         Object o = plus.invoke(obj,VOID);
         System.out.println("a+b="+((Integer)o));
      } catch(Exception e){
         System.out.println("exception!!! "+e);
         return;
      }
```

JavaBeans = Properties + EventSets + Methods

JavaBeans を実現する技術として、Java の reflection 機能を説明した。再度、JavaBeans の機能をまとめておくと、

1. Builder のデザインシートの上に、お絵描きソフトの感覚でクラスを張り付けることができる。つまり、Builder というツールが（外部の）Beans を操作することができる。
2. 属性（プロパティ）をプロパティシートに羅列された個々のプロパティエディタ上で変更し、保存できる。つまり、ロードした Beans の状態を変更することができる。
3. ２つのクラスに対して特定の「イベントオブジェクト」の交換を指定することで連携させることができる。
4. デザインシートごと保存して、実行可能な Java プログラムとすることができる。

reflection の機能は、１，２の機能を実現するために使われている。Beans は、JavaBeans の仕様で記述されたクラス（ファイル）である。基本的にはどのクラスファイルでも Beans として扱うことができるが、JavaBeans の仕様で書いておくことによって、Builder で内部の値を変更できるようになる。これを行うのが Java Beans の introspection 機能である。これは、reflection 機能を使って実装されており、高レベルの reflection 機能と言える。

JavaBeans は、JavaBeans の仕様に従ったクラスオブジェクトである。JavaBeans にある属性を持たせるには、JavaBeans の仕様に従ったメソッドを定義する。クラスの属性は大体はクラスのフィールド（インスタンス変数）で実現されることが多いのであるが、その属性をアクセスするメソッドの名前で決める。たとえば、ある Beans が foreground という名前のプロパティも持つのは、その beans が Color getForegroud()と void setForegroud(Color c)というメソッドを持つ場合である。プロパティをセットするメソッドを setter、取り出すメソッドを getter という。あるプロパティに対し、

setter= “set” + プロパティ名　　getter = “get” + プロパティ名

となる。通常プロパティ名は小文字になるが、１文字目と２文字目が両方とも大文字だったら、プロパティ名はそのままの名前を使うことに注意。

また、boolean プロパティの場合には、getter の名前は、”is”から始まる。

たとえば、

```
public class oneProp {
   private int p = 9;
   public int getProp() { return p; }
   public int setProp(int i) { p = i; }
}
```

では、prop という属性を持つ bean になる。

プロパティが変化した場合に、他の beans に伝える機能 PropertyChangeSupport クラスや、プロパティを変化させる PropertyEditor のクラスを指定する機能があるが、省略する。

さて、JavaBeans でもお互いのイベントを交換するために、AWT で解説した Event Listener モデルを使っている。これは、イベント発生する側にどこにそのイベントを伝えるかを設定し、イベントが伝えられる側にリスナーメソッドを定義するものである。ある JavaBeans が foo　という名前のイベントを発生するとする。このとき、beans はイベントセット foo を持つという。この beans は fooEvent というイベントオブジェクトの送り手とならなくてはならない。すなわち、この beans にこのイベントの受け手である FooListener を登録することができるメソッド addFooListener があることを意味する。javaBeans では、プロパティと同様に規則的な名前をつけることによって、定義する。

- Event オブジェクト　　class “イベント”Event　extends java.util.EventObject
- Listener　　interface “イベント”Listener extends java.util.EventListener
- Listerner 登録　　void add”イベント”Listener(“イベント”Listener listener)
- Listerner 抹消　　void remove”イベント”Listener(“イベント”Listener listener)

たとえば、次の例では一つのイベントセット foo をもつ beans である。

```
import java.util.*;
public interface FooListener extends { … }

import java.beans.*;
public class eventDesc {
  public void addFooListener(FooListener l){ … }
  public void removeFooListener(FooListener l) { … }
```

}
JavaBeans では、プロパティと同様に、このような名前の規則に従ったメソッドを探すことによって、イベントセットを見つける。
さて、Builder では以上にみるようなプロパティやイベントセット、メソッドを持っているかを調べて、それらを操作する。このような性質を調べる機能が introspection である。intropsector クラスを用いることによって調べることができるようになっている。

```
import java.beans.intropsector;
….
Beans Info = Intropsector.getBeanInfo(Class beanClass);
```

添付したソースプログラムは、この機能を利用してプロパティを調べるプログラムである。

Serialization

JavaBeans の全体像を明らかにする前に、Java の重要な機能である Serialization について述べる。この機能は、オブジェクトの状態をファイルやネットワーク上に書き出したり、読み込み復元したりする機能である。次の例は、Date というオブジェクトをファイルに書き出す例である。

```
Date d = new Date(); ….
FileOutputStream fout = new FileOutputStream(“tmp”);
ObjectOutputStream out = new ObjectOutputStream(fout);
out.write(d);
out.flush();
```

このファイルから、呼び出してオブジェクトを復元するには、

```
FileInputStream fin = new FileInputStream(“tmp”);
ObjectInputStream in = new ObjectInputStream(in);
Date d = (Date)in.readObject();
```

一つ注意することは、すべてのオブジェクトが writeObject で書き出すことができるわけではないことである。このように書き出すことのできるオブジェクトは、Serializable インタフェースを実装していなくてはならない。これをやるには単に、Serializable インタフェースをつければよい。

```
import java.io.Serializable;
public class myClass implements Serializable { … }
```

この機能は、Builder でいろいろな属性を変更したオブジェクトの状態をファイルにセーブし、プログラムの実行時に設定の状態を復元するのに用いられている。Seralization では、オブジェクト（つまりインスタンス）を転送している。例では、Date というクラスがすでにプログラムにあるが、未知のクラスのオブジェクトを復元するためにはそのクラスをロードしておかなくてはならない。
この機能はネットワークごしに他のマシンに任意のデータを転送したりする場合にも用いられている機能であり、Java の分散環境である Jini や RMI では重要な役割を果たす。

Beans は、JavaBeans の仕様で記述されたクラス（ファイル）である。JavaBeans の仕様で書いておくことによって、Builder で内部の値を変更したり、どのようなイベントを生成するか（イベントセット）を知ることができる。これを行うのが Java Beans の introspection 機能である。

Wiring とは

さて、JavaBeans でのプログラミングの中心となるのが、Wiring、すなわちそれぞれの JavaBeans のイベントを通じての関連づけをする部分である。Builder 内で、部品 A と部品 B について、マウスをつかって結びつける操作を行う。すなわち、この操作では部品 A で発生したイベントを B につたえて、部品 B のメソッドを呼び出すようにする。ここで、Java の GUI のプログラミングについて思い出してみよう。ボタン buttonA が actionEvent　　というイベントセットをもち、それを HelloLabel がうけとって、”hello”というメッセージを表示する場合、おこなわなくてはならないことは、HelloLabel を ActionListener としてインタフェースをつくっておき、これを buttonA のリスナーとして登録することである。

```
class buttonA { …
  buttonA(HelloLabel label) {
     … addActionListener(label); …
  } }
class HelloLabel implements ActionListener { …
   actionPerformed(ActionEvent ev) {  … showHello(); /*“hello”を表示*/ }
```

```
}
class Exec {
  public static void main(String argv[]) {
    HelloLabel l = new HelloLabel();
    new buttonA(l):
}
```
ここで、Exec は初期設定をするための main を持つクラスである。しかし、このようなプログラムをするには、プログラムの中にイベント処理のためのコードが埋め込まれてしまっている。このようなコードが埋め込まれていては、「部品」として他の用途につかうことができないということになってしまう。

Adapter クラスの利用

このような問題に対処するためには、Adapter というものをつかう。

```
class buttonA { … }
class HelloLabel { … }
class Adapter implements ActionListener {
  private HelloLabel target;
  public void setTarget(HelloLabel t){ target = t; }
  public void actionPerformed(ActionEvent ev) {  target.showHello(); }
}
class Exec {
   public static void main(String argv[]){
     Hello l = new HelloLabel();
     buttonA botton = new buttonA();
     Adapter adapter = new Adapter();
     adapter.setTarget(l):
     button.addActionListener(adapter);
}
```
つまり、イベントを受け取るための Adapter という仲介をするオブジェクトをつくって、イベントの橋渡しをさせることによって、元のコードにリスナーを作らなくてもよくなる。Exec.main ではこの adapter をつくってこれを Listener にしている。（実際には exec.main はない）

Builder での Wiring の処理

さて、このような動作を Builder ではどうしているのか。簡単な Builder である BeanBox では、Wiring に対して以下の処理を行っている。

1. Adapter クラスのプログラムを BeanBox の中で実行中に生成し、コンパイルして、それを動的にロードして実行している。
2. さらに、BeanBox で adaptor に target を設定したり、adaptor を Listener として設定したりするには reflection 機能を利用してクラスのメソッドを呼び出している。

１に関しては、beansbox/tmp/sun/beanbox/という directory に、__Hookup_????という名前のクラスのプログラムを作り、これが adaptor になっている。これをコンパイルし、できたクラスファイルを動的にロードする。２に関しては、以前 reflection のところで説明したとおり、java.lang.reflect の Method クラスに定義されている invoke メソッドをつかって、setTarget や addActionListener を呼び出せばよい。

Applet の生成

BeanBox では適当に部品を配置し、wiring してできたプログラムを Applet として save することができる。これによって、たとえば、applet の名前を MyApplet とすると、所定の directory (./tmp/MyApplet) に、以下のファイルが作られる。

1. MyApplet.html: applet をテストするための HTML ファイル
2. MyApplet_files: ここには生成された applet のプログラムと data が入る。
3. MyApplet.jar: MyApplet_files を JAR にしてまとめたもの。
4. その他の必要な beans の入った JAR ファイル

JAR ファイルは複数のクラスをまとめたファイルである。テスト用の HTML ファイルには、以下のように記述されている。

```
<applet
   archive="./myApplet.jar,./support.jar
```

```
      ,./mytools.jar
   "
   code="Hello"
   width=390
   height=492
>
```
通常、Applet には、codebase=として、クラスファイルのある web サーバ側の directory 指定するが、ここでは archive として必要な JAR ファイルを指定している。

生成された Applet のプログラムは、MyApplet_files/Hello.java にある。通常のアプレットとは違って、コンストラクタのところでほとんどの処理をしている。ここから呼ばれる initContents では、Beans.instantiate は new の働きをするのであるが、これはもしも”OurButton”という名前で、serialization されたオブジェクトがあった場合、そのオブジェクトを元にインスタンスを作る操作をするメソッドである。ここでは、new と同じである。その後、BeansBox で設定したプロパティをセットしている。

次に呼ばれているのは、addConnections である。ここでは、_hookup_???として作った Adaptor を生成し、実際にプログラムのなかで、target をセットし、Listener として登録するコードを出している。もはや、BeanBox ではないので、実際のコードを出力すればいいことに注意。

その他のメソッドとして、この applet をオブジェクトとして書き出したり、読み込んだりするメソッド readObject/writeObject がある。

JAR ファイルと Bean の作り方

java のクラスのファイルは、jar というコマンドで JAR ファイルにしておくことができる。実際のクラスのある directory に対するパスを設定する代わりに、このファイルをクラスパスに設定しておけばこの中にあるクラスファイルが参照されるようになる。jar ファイルの作り方は、tar に似ている。クラスのある directory で、

% jar –cvf　directory

とすればよい。

実際、jar のファイル形式は、zip 形式を使ったものなので、zip/unzip コマンドでも作ることができる。自分が作った Bean を BeanBox などの Builer にもっていくためには JAR ファイルにしておかなくてはならない。但し、単なる JAR ファイルではなくて、

```
Name: クラスファイル名
Java-Bean: true

Name: クラスファイル名
Java-Bean: true
```
…

というどれが Bean であるかというファイルを作って、これを manifest として作らなくてはならない、このファイルを manifest.tmp とすると、

%jar cfm JARFILE manifest.tp *.class

として作る。m は manifest ファイルを指定するオプションである。これをクラスパスに設定しておけば、builder で使えるようになる。（但し、beanBox では、../jars しかみていないようだ）