7.5 PDF による帳票出力

iText

業務アプリケーションをウェブで作ろうとすると、必ずネックになるのが「帳票をどうやって出力するか」だ。

一番簡単な方法は“HTML で表を組んで、そのままブラウザから印刷してしまう”というものだ。スタイルシートをうまく使えば、それなりに見栄えのする帳票ができないことも無い。ただ、余白の設定やヘッダ、フッタなど、どうしてもブラウザの設定に依存する部分が残ってしまうのだ。MSIE では ScriptX というスクリプトを使ってブラウザの設定を強制的に変更することもできるようだが、そのためにはクライアント側に ScriptX をインストールしておく必要があったりして、今ひとつメジャーな解決策になりきれてない感じだ。

こういうところに使えそうなのが iText というライブラリだ。iText はプログラムから動的に PDF データを生成するための、かなり大がかりなパッケージで。RTF や HTML、XML にも対応しているらしい。ここではその一部の機能を使って帳票を出力する。

> iText は Bruno Lowagie 氏と Paulo Soares 氏によるオープンソースのプロジェクトである。最新版は http://www.lowagie.com/iText/から入手可能だ。

iText のインストール

インストールと言っても Java のパッケージだから iText.jar ファイルを CLASS_PATH の通っているディレクトリに置くだけだ。たとえば、これを Java のディレクトリの jre の下の lib の下の ext に入れておく（Windows の場合は 2 か所）。日本語を扱う場合は iTextAsian.jar も必要になるので、同じディレクトリに入れておく。

簡単な PDF を表示するサーブレット

できれば JSP でやってみたかったのだが、惜しいところでうまくいかなかった（悔しい）。ちょっと面倒だが、サーブレットでやってみよう。ソースはこんな感じだ。

/CD-ROM/<OS>/SRC/7.05/SimplePDF.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import com.lowagie.text.*;                  //iText 拡張 API
import com.lowagie.text.pdf.*;              //iText 拡張 API
```

その他の小技

```
public class SimplePDF extends HttpServlet{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
               throws ServletException, IOException{
       //ByteArrayOutputStreamの準備
       ByteArrayOutputStream baos = new ByteArrayOutputStream();
       //Documentの作成
       Document document = new Document(PageSize.A4, 50, 50, 50, 50);
       try{
            //PdfWriterの作成
            PdfWriter pdfwriter = PdfWriter.getInstance(document,baos);
            document.open();
            //BaseFontの作成
            BaseFont bf = BaseFont.createFont(
                          "HeiseiKakuGo-W5", "UniJIS-UCS2-H",false);
            //PdfContentByteの作成
            PdfContentByte pcb = pdfwriter.getDirectContent();
            //ブロック1
            pcb.beginText();
            pcb.setTextMatrix(50, 800);    //座標の設定
            pcb.setFontAndSize(bf, 12);    //フォントの設定
            pcb.setLeading(15);            //改行幅の設定
            pcb.showText("げっ");          //文字列の書き出し
            pcb.showText("ちゅー");
            pcb.newlineText();
            pcb.showText("ありおりはべり");
            pcb.newlineShowText("いまそかり");
            pcb.endText();
            //ブロック2
            pcb.beginText();
            pcb.setFontAndSize(bf, 30);
            //テキストのセンタリング
            pcb.showTextAligned(PdfContentByte.ALIGN_CENTER,"がちょ〜ん",297,750,0);
            pcb.endText();

       }catch(Exception e){
            e.printStackTrace();
       }
       document.close();
       //レスポンスヘッダの設定
       response.setContentType("application/pdf");
       response.setContentLength(baos.size());
       //ブラウザへ送信
       ServletOutputStream out = response.getOutputStream();
       baos.writeTo(out);
       out.flush();
   }
}
```

javax.servlet と javax.servlet.http はサーブレットを書くときに必須のパッケージだ

java.io は ByteArrayOutputStream を使うためにインポートする。そして、com.lowagie.text と com.lowagie.text.pdf が iText のパッケージだ。

ByteArrayOutputStream の準備

ByteArrayOutputStream は読んで字のごとく、バイト配列を出力するためのストリームだ。iText の PdfWriter というオブジェクトが書き出す PDF データを、一度このストリームに溜めておいて、最後にブラウザへ送信する。

Document の作成

Document は iText が扱う“文書”という抽象的なものだ。コンストラクタで文書のサイズと余白を指定し、オブジェクトを作ったらそれを、開いて、書いて、閉じる。Document クラスは数多くのメソッドを持っているが、ここでは getInstance()と open()と close()しか使っていない。余白の単位はポイントだ。デフォルトでは 72dpi になっている。つまり、1 ポイントは 25.4mm/72=約 0.35mm だ。A4 サイズの紙なら横が 595 ポイント、縦が 842 ポイントということになる。

PdfWriter の作成

PdfWriter の getInstance()メソッドに Document と ByteArrayOutputStream を渡してインスタンスを作っている。pdf データをファイルに書き出す場合は、ByteArrayOutputStream の代わりに FileOutputStream を指定すればいいらしい。

BaseFont の作成

BaseFont はフォントの種類を指定するためのクラスだ。createFont メソッドでフォント名、エンコーディングタイプ、組み込みの有無を指定する。日本語の場合、フォント名にはゴシック系の HeiseiKakuGo-W5 または明朝系の HeiseiMin-W3 が指定できる（それしか指定できない）。エンコーディングは UniJIS-UCS2-H 、 UniJIS-UCS2-V 、 UniJIS-UCS2-HW-H 、UniJIS-UCS2-HW-V のいずれかを指定する。-V は縦書き用。HW が付いているのは半角文字が半角（って書くと変だが、要するにプロポーショナルではないということ）。日本語フォントは組み込みにできないので、3 つめのパラメータは false にする。

PdfContentByte の作成

PdfWriter の getDirectContent()メソッドで PdfContentByte というオブジェクトを取得している。これは、テキストブロック、もしくは、レイアウト枠のようなものだと思えばいい。beginText()と endText()の間に出力したものが、ひとつのまとまりとして扱われる。

座標・フォント・改行幅の設定

setTextMatrix()メソッドでは印字開始の場所を指定している。出力する紙（Document）の右

その他の小技

下が座標の原点(0,0)で、右上の座標が(595,842)ぐらいになる。setFontAndSize()でフォントの種類とサイズを指定している。setLeading()では改行幅を指定している。

文字列の書き出し

PdfContentByte への出力のしかたにはいろいろあるが、この例では、showText()とnewlineShowText()を使っている。違いは文字を出力する前に改行するかどうかだけだ。改行だなら newlineText()を使えばいい。endText()で一度テキストブロックを終わらせ、beginText()で次のテキストブロックを始めている。

センタリングと右詰め

2 つめのテキストブロックでは、showTextAligned()メソッドを使っている。テキストをセンタリングしたり、右詰めにしたい場合に使うメソッドだ。最初のパラメータで左詰、センタリング、右詰めのいずれかを指定する。この指定のために PdfContentByte クラスで次のような定数が定義されている。

定数	意味	値
ALIGN_CENTER	中央揃え	1
ALIGN_LEFT	左詰	0
ALIGN_RIGHT	右詰	2

“1”と書けば済むところを、わざわざ定数を使って PdfContentByte.ALIGN_CENTER と書くのはかなり面倒な感じだが、これでソースが読みやすくなるのだから我慢した方がいい。

レスポンスヘッダの設定

ここでブラウザに返すレスポンスヘッダを設定する。通常のウェブコンテンツは ContentType が "text/html" だが、PDF データを返す場合は"application/pdf"とする。これを受信したブラウザは、PDF を表示できるアプリケーションを用意するわけだ。ほとんどの場合は Adobe の Acrobat か AcrobatReader だが、これらがインストールされていない場合は「保存しますか？」のダイアログが表示される。

ContentLength はブラウザが受信完了を判断する目安にするためのヘッダで、ByteArrayOutputStream の size()メソッドで取り出した値を指定している。

response.getOutputStream() メソッドで PDF データを送り出すための ServletOutputStream を取得している。

ブラウザへの送信

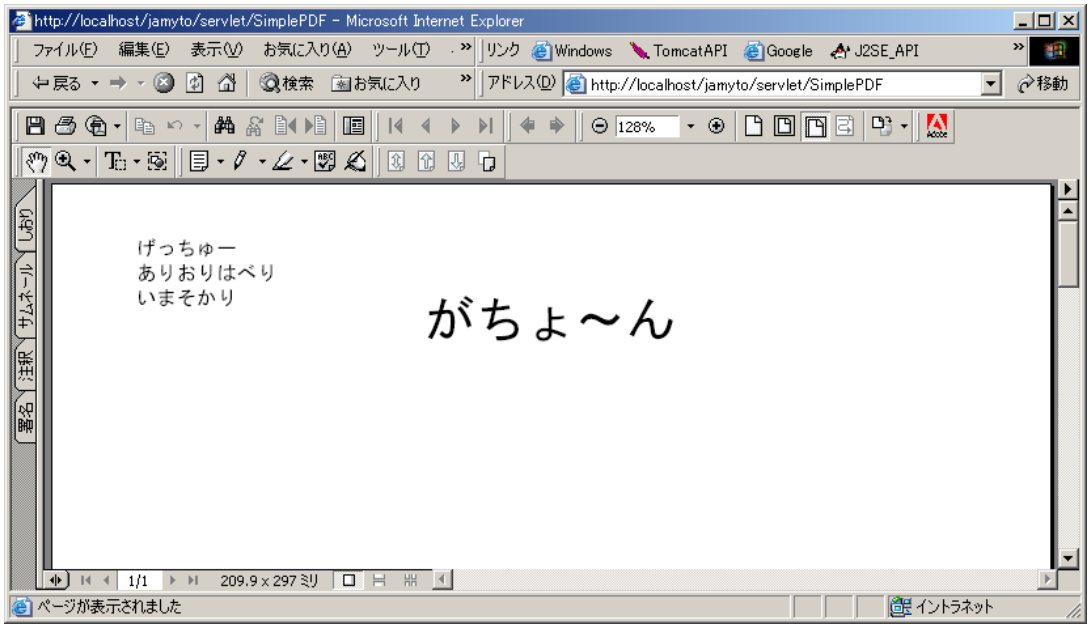
現在ByteArrayOutputStreamに入っているPDFデータをServletOutputStreamに書き出し、

最後に ServletOutputStream の内容をブラウザに送り出している。

コンパイルと実行

サーブレットをコンパイルする場合は、Tomcat の common/lib にある servlet.jar にクラスパスを通しておく必要がある。javac のコマンドラインで-classpath を使って指定してもいいが、シェルの環境変数で設定しておくと面倒がない。

コンパイルしてできた SimpePDF.class を jamyto/WEB-INF/classes にコピーし、ブラウザから http://localhost/jamyto/servlet/SimplePDF にアクセスする。AcrobatReader がインストールされていれば、ブラウザの中に動的に生成された PDF 文書が表示される。こんな感じだ。



動的に生成された PDF 文書

表組み

帳票を印刷しようとしたら表組みは避けて通れない。だが、iText のテーブルはちょっと大変だ。

/CD-ROM/<OS>/SRC/7.05/SimplePDFTable.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
import java.text.*;
import com.lowagie.text.*;                  //iText 拡張 API
import com.lowagie.text.pdf.*;              //iText 拡張 API
```

```
public class SimplePDFTable extends HttpServlet{
   public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, IOException{

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        Document document = new Document(PageSize.A4, 50, 50, 50, 50);

        /*ドキュメントの内容*/
        try{
            //出力先の設定
            PdfWriter pdfwriter = PdfWriter.getInstance(document,baos);
            document.open();
            BaseFont bf = BaseFont.createFont(
                "HeiseiKakuGo-W5", "UniJIS-UCS2-H",false);
            PdfContentByte pcb = pdfwriter.getDirectContent();

            //テーブルの作成
            PdfPTable table = new PdfPTable(3);
            //テーブルの幅を指定
            table.setTotalWidth(300);
            int width[]={50,30,20};
            table.setWidths(width);
            //罫線の設定
            table.getDefaultCell().setBorder(Rectangle.BOX);
            //フォントの設定
            Font font=new Font(bf,12);
            PdfPCell cell;

            cell = new PdfPCell(new Phrase("セル1",font));   //追加するセルを作成
            table.addCell(cell);                             //セルをテーブルに追加

            cell = new PdfPCell(new Phrase("セル2",font));
            table.addCell(cell);

            cell = new PdfPCell(new Phrase("セル3",font));
            table.addCell(cell);

            cell = new PdfPCell(new Phrase("セル4",font));
            cell.setColspan(2);
            cell.setHorizontalAlignment(Element.ALIGN_CENTER);
            table.addCell(cell);

            cell = new PdfPCell(new Phrase("セル5",font));
            cell.setHorizontalAlignment(Element.ALIGN_RIGHT);
            table.addCell(cell);
            //テーブルをドキュメントに追加
            table.writeSelectedRows(0, -1, 100, 600, pdfwriter.getDirectContent());

        }catch(Exception e){
            e.printStackTrace();
```

```
        }

        //ドキュメントを閉じる
        document.close();

        //クライアントに応答
        response.setContentType("application/pdf");
        response.setContentLength(baos.size());
        ServletOutputStream out = response.getOutputStream();
        baos.writeTo(out);
        out.flush();
    }
}
```

表の出力に関する部分は次のようなコードである。

テーブルの設定

PdfPTable のインスタンスを作っている。パラメータは 1 行に入るセルの数だ（列とも言う）。setTotalWidth()でテーブルの幅をポイントで指定し、setWidth()では配列を使って個々のカラムの幅を設定している。値は相対値だ。

罫線の設定

getDefaultCell().setBorder()でデフォルトのセルに対して罫線を指定している。setBorder()メソッドの引数は Rectangle クラスで次のように定義されている定数を使う。

定数	意味	値
BOTTOM	底辺	2
BOX	四角	15
LEFT	左辺	4
NO_BORDER	罫線なし	0
RIGHT	右辺	8
TOP	上辺	1

罫線をセルの上と下に引きたい場合はビット演算子の OR（|）を使って(Rectangle.TOP|Rectangle.BOTTOM)のように指定する。

追加するセルを作成する

ここからは少し複雑だ。まず、PdfPCell 型のテンポラリなオブジェクト cell を作る。次に追加するセルの内容を cell オブジェクトに設定する。セルの内容を設定するには、Phrase 型のオブジェクトを使って、

```
cell=newPdfPCell(new Phrase("内容",font));
```

のようにする。

セルをテーブルに追加する

作ったセルをテーブルに追加する。追加したセルは、テーブルの左上から順に並べられ、テーブル作成時に指定された列の数（この例では 3）を越えると次の行へ折りかえす。

PdfPCell の setColspan()メソッドで 1 つのセルが 2 つ分の幅を持つように設定している。HTML の<TD COLSPAN="2"> と同じような設定方法だ。

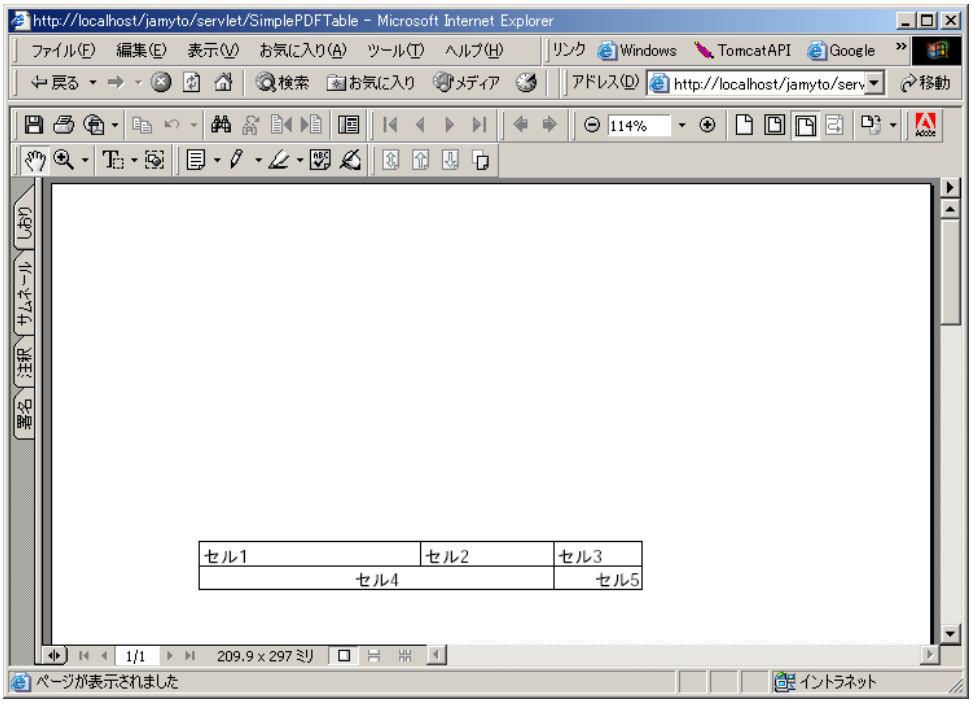
PdfPCell の setHorizontalAlignment()メソッドを使ってセル内の文字をセンタリングしている。ここで使っている Element クラスの定数は次のようなものだ。

定数	意味	値
ALIGN_LEFT	左詰め	0
ALIGN_CENTER	中央（左右）	1
ALIGN_RIGHT	右詰め	2
ALIGN_JUSTIFIED	両端揃え	3
ALIGN_TOP	上詰め	4
ALIGN_MIDDLE	中央（上下）	5
ALIGN_BOTTOM	下詰め	6
ALIGN_BASELINE	ベースライン	7

テーブルをドキュメントに追加する

テーブルをドキュメントに追加するのに PdfPTable オブジェクトの writeSelectedRows()メソッドを使っている。document.add(table)でもテーブルを書き出すことができるが、このメソッドを使えば、テーブルを書き出す位置を指定できるのだ。writeSelectedRows()メソッドのパラメータは開始行、終了行、書き出し位置 X、書き出し位置 Y の順だ。行の指定は、最初の行が 0 となる。また、終了行に-1 を指定すると、すべての行が書き出される。書き出し位置の X と Y は左下からのポイント数だ。

表示はこんなふうになる。



表組み

グラフィックの描画

iText の PdfContentByte クラスには図形を描画するためのメソッドが数多く備えられている。その中のいくつかを使って直線と四角形の描画、イメージの貼り付けをやってみた。ソースはこんな感じだ。

/CD-ROM/<OS>/SRC/7.05/SimplePDFDraw.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
import java.text.*;
import com.lowagie.text.*;                  //iText 拡張 API
import com.lowagie.text.pdf.*;              //iText 拡張 API

public class SimplePDFDraw extends HttpServlet{
   public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, IOException{
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        Document document = new Document(PageSize.A4, 50, 50, 50, 50);

        /*ドキュメントの内容*/
        try{
```

その他の小技

```
            //出力先の設定
            PdfWriter pdfwriter = PdfWriter.getInstance(document,baos);
            document.open();
            BaseFont bf = BaseFont.createFont(
                "HeiseiKakuGo-W5", "UniJIS-UCS2-H",false);
            PdfContentByte pcb = pdfwriter.getDirectContent();

            pcb.setLineWidth(2.8F);        //線の太さを指定
            pcb.moveTo(30,800);            //線の始点
            pcb.lineTo(565,800);           //線の終点
            pcb.stroke();         //線を引く

            pcb.moveTo(50,750);            //連続した線の描画
            pcb.lineTo(70,780);
            pcb.lineTo(150,720);
            pcb.stroke();

            pcb.rectangle(200,650,80,80);  //四角形の描画
            pcb.stroke();

            Image img=Image.getInstance(   //ビットマップの貼り付け
                getServletContext().getRealPath("img")+"/getyou.png");
            img.setAbsolutePosition(230,630);
            pcb.addImage(img);

            pcb.rectangle(300,600,80,80);
            pcb.stroke();

        }catch(Exception e){
            e.printStackTrace();
        }

        //ドキュメントを閉じる
        document.close();

        //クライアントに応答
        response.setContentType("application/pdf");
        response.setContentLength(baos.size());
        ServletOutputStream out = response.getOutputStream();
        baos.writeTo(out);
        out.flush();
    }
}
```

線の太さ

まず `setLineWidth()`メソッドを使って線の太さを設定する。パラメータはドット数だが、型が `float` なので数字に `f` をつけている。

線の描画

線を引く場合は、moveTo()メソッドで始点を指定し、次に lineTo()メソッドで終点を指定して stroke()メソッドを呼び出す。lineTo()メソッドで複数の点を指定すると、これらを結ぶ折れ線を引くことができる。

四角形の描画

四角形を書く場合は rectangle()メソッドで左下の角の X 座標、Y 座標、幅、高さの順にパラメータを指定し、stroke()メソッドを呼び出す。

ビットマップの貼り付け

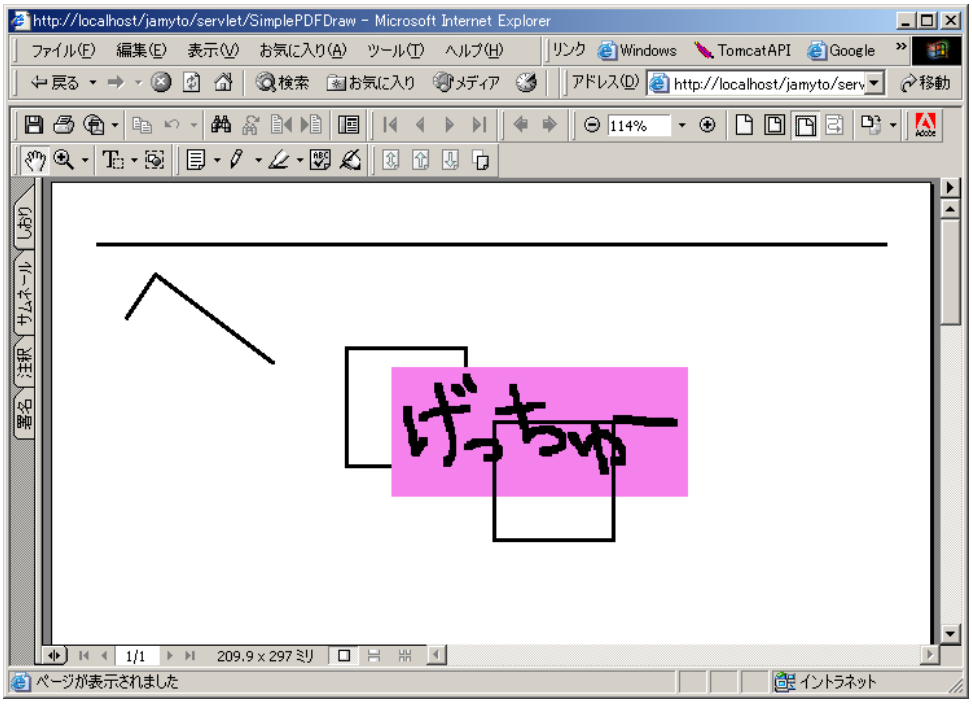
ビットマップイメージを貼り付ける場合は、まず Image クラスの getInstance()メソッドでオブジェクトを作る（Image クラスは iText パッケージに含まれている）。getInstance()メソッドのパラメータはファイル名または URL が指定できる。ファイル名を指定する場合は、getServletContext().getRealPath()でイメージファイルの入っているディレクトリ名を取ってくるといい。上のコードでは jamyto/img からイメージファイルを取ってくる。

URL を指定する場合は getInstance(new URL("http://～"))という感じだ。URL クラスは java.net パッケージののもを使うので、これを import すること。

コンパイルと実行

このコードをコンパイルして実行すると、こんな画面になる。

その他の小技



グラフィックの描画

これを見る限り、後から書いたものが手前に表示されるようだ。