NAME

XPDFJ - 日本語 PDF 生成モジュール PDFJ の XML フロントエンド

SYNOPSIS

```
use XPDFJ;
$xpdfj = new XPDFJ;
$xpdfj->parsefile($xmlfile, outfile => $pdffile);
```

DESCRIPTION

■概要

XPDFJ は、XML 形式で書かれた原稿から PDFJ を用いて PDF を生成する。
XML の要素が PDFJ のサブルーチンやメソッドの呼び出しに直接変換されるほか、実行結果を変数に格納して後で利用したり、補助的に Perl スクリプトを埋め込むこともできるので、PDFJ を使って直接スクリプトを書くのに近い細かな制御がおこなえる。
また新たなタグを定義できるマクロ機能もあり、 <UL><OL><LI><TABLE><TR><TD><HR> などの HTML ライクなタグを定義したファイルが用意されている。

■使用法

以下で説明する規則に従って原稿となる XML データを用意し、

```
$xpdfj = new XPDFJ;
```

として XPDFJ オブジェクトを作成し、

```
$xpdfj->parse($xmldata);
$xpdfj->parsefile($xmlfile);
```

のいずれかの方法で、XML データを処理する。

```
$xpdfj->parsefile($xmlfile, outfile => $outfile);
```

のようにハッシュリストの形で追加の引数を与えると、その XML データを実行するときに %Args というハッシュにセットされる。この例のようにしておいて、XML データで <print file="$Args{outfile}"/> とすれば、生成された PDF が $outfile に出力されることになる。

```
$xpdfj = new XPDFJ(dopath => '/usr/local/XPDFJ;/usr/local/PDFJ');
```

のように、XPDFJ オブジェクトを作成するときに dopath という引数を与えると、<do file> するときにそのパスからファイルが検索される。セミコロン区切りであることに注意。

```
$xpdfj = new XPDFJ(debuginfo => 1, verbose => 2);
```

のように debuginfo を真に指定すると、後述する <debuginfo> が有効になる。verbose を 1 または 2 にすると内部的な実行状況が表示される。
debuginfo と verbose は、

```
$xpdfj->debuginfo(1);
$xpdfj->verbose(2);
```

のようにして必要なときにセットすることもできる。

■ XML データの基本構成

XPDFJ の原稿となる XML データは、整形式 XML であり、ルート要素は XPDFJ でなければならない。

```
<?xml version="1.0" encoding="x-sjis-cp932"?>
<XPDFJ version="0.1">
…
</XPDFJ>
```

XML データは XML::Parser モジュールによって読み取られる。日本語を含む場合は上記の例のように先頭の XML 宣言で文字コードに応じた適切なエンコーディングを指定しなければならない。
<XPDFJ version="0.1"> とバージョンを指定すると、XPDFJ のバージョンがそれよりも小さければエラーとなって実行できない。
XPDFJ では、ある XML データを処理中に、その中で指定された他の XML ファイルを読み込む機能がある。この機能によって読み取られる XML ファイルの場合は、整形式であればよく、ルート要素は XPDFJ でなくてよい。
※今のところ、XPDFJ 用の DTD などは用意されていない。

■ XML 要素の扱い

XML データ中の各要素およびテキストは、次の規則に従って処理される。
※以下の説明で、「ワード」は半角の英字またはアンダースコアではじまり、英数字またはアンダースコアからなる文字列のことを意味する。

- 空白文字だけからなり改行を含むテキストはすべて無視される
- ルート要素である XPDFJ 要素の内容は、順にメソッドモードで処理される
- メソッドモードでは、テキストは無視される
- メソッドモードでは、要素は、要素名がマクロ定義されたタグに合致すればその定義にしたがって処理され、内部関数名に合致すれば内部関数として、大文字で始まるワードであればサブルーチンとして、小文字で始まるワードであればメソッドとして扱われ、実行される。そしてその実行結果のリストが返される
- 要素がマクロ定義されたものの場合には、その属性や内容の扱いはその定義による（下記のマクロ定義の説明を参照）
- 要素が内部関数として扱われる場合には、その属性や内容の扱いは内部関数による（下記の各内部関数の説明を参照）
- 要素がサブルーチンまたはメソッドとして扱われる場合には、属性はそのまま、内容は引数モードで処理されて、一つのハッシュにまとめられ、そのハッシュ参照がサブルーチンまたはメソッドの引数として与えられて実行される（いくつかある特殊扱いの引数については後述）
- 要素がメソッドとして扱われる場合、呼び出し元のオブジェクトは通常は Doc オブジェクトとなる（それ以外のオブジェクトを指定する方法は後述）
- 引数モードでは、テキストは無視される

・ 引数モードでは、要素は、要素名が引数名となり、内容がテキストのみならその文字列が引数値となり、そうでなければ内容をメソッドモードで順次実行した実行結果のリストが引数値となる。（リストが１要素の場合はその値、２要素以上の場合は配列参照として）引数値が文字列で、それがスカラ変数名やハッシュ要素名であればその値に置き換えられる

この説明だけではわかりにくいので、例をあげる。

```
<XPDFJ version="0.1">
  <Doc version="1.3" pagewidth="595" pageheight="842"/>
  <Text>
    <texts> テキスト </texts>
    <texts>text</texts>
    <style>
      <TStyle fontsize="10">
        <font>
          <new_font basefont="Ryumin-Light" encoding="UniJIS-UCS2-HW-H"/>
        </font>
      </TStyle>
    </style>
  </Text>
</XPDFJ>
```

XPDFJ ルート要素の内容は（テキストは無視されて）<Doc> と <Text> である。これがメソッドモードで順次処理される。
<Doc> は空要素なので、属性がそのまま引数となり、Doc({version => "1.3", pagewidth => "595", pageheight => "842"}); と実行される
<Text> は属性がないので、内容である <texts><texts><style> が引数モードで処理されて引数となる。
<texts> は二つあっていずれもテキストのみが内容なので、その内容のリストの配列参照が値となる。
<style> の内容は <TStyle> なのでその実行結果、つまりテキストスタイルオブジェクトが値となる。
<new_font> は小文字で始まるのでメソッド呼び出しであり、呼び出し元メソッドは Doc オブジェクトとなる。
結局上記の XML データを処理すると、次のスクリプトが実行されるのと同じことになる。

```
$Doc = Doc({version => 1.3, pagewidth => 595, pageheight => 842});
Text({
  texts => [' テキスト ', 'text'],
  style => TStyle({
    fontsize => 10,
    font => $Doc->new_font({basefont => 'Ryumin-Light',
      encoding => 'UniJIS-UCS2-HW-H'})
    })
  });
```

■特殊扱いの引数

要素がサブルーチンまたはメソッドとして実行されるとき、次の引数は特別な扱いとなり、サブルーチンまたはメソッドの引数としては渡されない。

setvar

サブルーチンまたはメソッドの実行結果を格納する変数名を指定する。変数名としては、スカラ変数名またはハッシュ要素名が指定できる。

```
  <new_font setvar="$Font{mincho}" basefont="Ryumin-Light" encoding="UniJIS-UCS2
-HW-H"/>
  <TStyle setvar="$TStyle{normal}" font="$Font{mincho}" fontsize="10"/>
  <Text setvar="$text1" texts="テキスト" style="$TStyle{normal}"/>
```

のようなことができる。

call

サブルーチンまたはメソッドの実行結果を呼び出し元オブジェクトとして実行したい内容を指定する。

```
  <PStyle setvar="$PStyle{normal}" size="200" align="w" linefeed="150%">
    <call>
      <clone setvar="$PStyle{indent}" begginpadding="30"/>
      <clone setvar="$PStyle{note}" labeltext="※" labelsize="10"/>
    </call>
  </PStyle>
```

二つの <clone> は普通であれば Doc オブジェクトを呼び出し元オブジェクトとして実行されるが、ここでは <call> の内容となっているので、その親である <PStyle> の実行結果である段落スタイルオブジェクトを呼び出し元として実行される。
サブルーチンまたはメソッドの実行結果がリストであれば、その各要素を順次呼び出し元として繰り返し実行される。

```
  <break caller="$blockobj" sizes="200">
    <call>
      <new_page caller="$Doc" setvar="$page"/>
      <show page="$page" x="10" y="10" align="bl"/>
    </call>
  </break>
```

この例はブロックオブジェクト $blockobj をサイズ 200 で break し、その結果の各ブロックオブジェクトを新たなページに配置している。

caller

メソッドの場合のみ特別扱いとなり、呼び出し元オブジェクトを指定する。
上記の例は次のように書くこともできる。

```
  <PStyle setvar="$PStyle{normal}" size="200" align="w" linefeed="150%"/>
  <clone setvar="$PStyle{indent}" caller="$PStyle{normal}" begginpadding="30"/>
  <clone setvar="$PStyle{note}" caller="$PStyle{normal}" labeltext="※" labelsize="10
"/>
```

attributes

attributes 属性にハッシュ参照を与えると、そのハッシュの内容が属性として扱われる。次の例のように、属性をまとめて変数にセットしておいて与えることができる。

```
  <eval>$attr = {size => 200, align => 'w', linefeed => '150%'}</eval>
  <PStyle setvar="$pstyle" attributes="$attr"/>
```

※暗黙のメソッド呼び出し元が Doc オブジェクトになるのは、 <Doc> の呼び出し時に自動的に setvar="$Doc" が付加され、メソッドで caller が省略されると caller="$Doc" となる、という仕組みになっている。

■変数

変数は、Perl のスカラ変数かハッシュ要素のみである。つまり、$variable か、$Hash{key} の形だけが変数とみなされる。また、変数を属性値や要素の内容として指定するときは前後も含めて余計な空白を入れると変数として扱われないので注意すること。（この制約は将来緩和されるかもしれない。）
変数は後述する <eval> や <reval> の中の Perl コードで操作することも可能である。
%Args というハッシュは、マクロ定義されたタグが実行される時や、追加の属性のある <do> の実行の際に、属性の名前と値が自動的にセットされる。そのタグや <do> の実行が終わると %Args の状態は実行前の状態に戻る。
%Args 以外の変数は全てグローバルである。局所化したい場合は、ローカルモード（後述）内で <local> を用いる。

■内部関数

XPDFJ に対する指示を表す内部関数には次のものがある。

<require>
<require XPDFJ="0.12"/> のようにすれば、XPDFJ のバージョンが指定した値以上であるかどうかをチェックする。

<eval>
内容を Perl のコードとして実行する。変数の状態はその XML データを処理している XPDFJ オブジェクトが生きている間は保持されている。use や require など外部を呼び出すコードは実行できない。print は可能なので、進行状況を表示したりできる。
<eval> 自体は何も結果を返さない。

```
<eval>
  $PStyle{default} = $PStyle{normal};
  $counter++;
  print "counter: $counter\n";
</eval>
```

<reval>
<eval> と同様に Perl のコードを実行するが、 <reval> はその結果を返す。

<for>
属性 eval の値を評価した結果のリストまたは配列参照の各要素を、属性 setvar で指定された変数に順次入れながら、内容を繰り返す。

```
<for setvar="$var" eval="(1, 2, 3)">
  <eval>print "$var\n"</eval>
</for>
```

<for> 自体は何も結果を返さない。

<do>

属性で与えたさまざまな条件に従って内容を実行する。指定できる属性が多岐に渡るので、下記の別項で説明する。

<local>

変数値を局所化する。setvar で指定された変数の値を保存し、eval が指定されていればその評価結果をセットする。保存された値は、ローカルモードの <do> やマクロ定義されたタグを抜ける時に元に戻される。

```
<local setvar="$counter" eval="1"/>
```

<local> 自体は何も結果を返さない。

<sub>

その内容を実行するサブルーチン参照を返す。PDFJ では段落スタイルの labeltext にサブルーチン参照を与えると、毎回それを実行した結果のテキストオブジェクトをラベルとする、という機能がある。このために使用する。

```
<sub>
  <Text style="$s_normal">
    <texts>$counter</texts><texts>.</texts>
  </Text>
  <eval>$counter++</eval>
</sub>
```

<def>

新たなタグを定義するマクロ定義命令。 <def> の機能は複雑なので、下記の別項で説明する。

<alias>

要素名に別名を付けるマクロ定義命令。tag 属性で指定した名前を、aliasof 属性で指定した名前の別名とする。それ以外の属性を与えると、追加の属性となる。

```
<alias tag="BR" aliasof="NewLine"/>
```

このように定義されていると、
 は <NewLine/> に置き換えられる。

```
<alias tag="TH" aliasof="TD" align="center"/>
```

このように定義されていると、 <TH> は <TD align="center"> に置き換えられる。
ただし alias で置き換えられるのは、メソッドモードでの要素名のみであり、引数モードでの要素名は対象とならない。
<alias> 自体は何も結果を返さない。

<debuginfo>

XPDFJ->new() の引数で debuginfo を真に指定していると、デバッグ用に現在の変数とその値の一覧を表示する。pattern 属性を指定すると、正規表現としてそのパターンにマッチする変数名のみが対象になる。

```
<debuginfo pattern="Args"/>
```

<debuginfo> 自体は何も結果を返さない。

■ <do> による実行

<do> は、属性で与えたさまざまな条件に従って内容を実行する。属性には次のものを指定できる。

if

その値を評価した結果が真なら実行する

```
<do if="$page->pagenum > 1"/>
  …
</do>
```

unless

その値を評価した結果が偽なら実行する

caller

省略時の呼び出し元オブジェクトを指定する

```
<do caller="$PStyle{normal}">
  <clone setvar="$PStyle{indent}" begginpadding="30"/>
  <clone setvar="$PStyle{note}" labeltext="※" labelsize="10"/>
</do>
```

setvar

実行結果をセットする変数名（次の result の指定にかかわらず、<do> の内容の実行結果がセットされる）

result

<do> 自体の返す値の指定：first: 実行結果リストの最初の要素、last: 実行結果リストの最後の要素、null: 何も返さない、変数名 : その変数値

メソッドモードで、変数にセットしたいだけで結果は必要ないときは、次のようにすればよい。

```
<do result="null">
  <new_font setvar="…/>
  <PStyle setvar="…/>
  <clone setvar="…/>
</do>
```

file

その値の外部ファイルを読み込んで内容として実行する（<do> 自体の内容は無視される）

```
<do file="stddefs.inc"/>
```

XPDFJ オブジェクトを作成するときに、dopath 引数を指定していれば、そのパスからファイルが検索される。

contents

その値を内容として実行する（<do> 自体の内容は無視される）。 <def> の contentsmode="raw" と組み合わせて使用する。

実行時の %Args は、実行される内容が書かれていた場所での %Args がベースとなる。

withtext

内容を実行するとき通常はテキストは無視されるが、withtext に真を指定するとテキストも結果リストに含まれる。withtext に「autonl」を指定すると、テキスト中の改行を NewLine オブジェクトに

置き換える

local
: local を真に指定すると、ローカルモードで内容を実行する（<local> を使って変数値を局所化できる）

verbose
: デバッグ用の進行状況表示フラグ（1 または 2）をセットする

以上のもの以外の属性を与えると、それは <do> の内容の実行開始時に %Args に追加セットされる。そして <do> の内容はローカルモードで実行され、%Args の内容は <do> の実行終了時に開始時の状態に戻される。

■ <def> によるマクロ定義

<def> は新たなタグを定義するマクロ定義命令。tag 属性で指定されたタグを定義する。定義されたタグが実行される時には、<def> の内容がローカルモードで実行される。実行開始時に <def> の属性と実行時の属性とが %Args に追加セットされ、%Args の内容は終了時に戻される。実行時の内容がどう扱われるかは、下記の contentsmode 指定による。
<def> の属性の値に変数を指定すると実行時に評価される。 <def> の属性と同名の属性を実行時に指定すると実行時の属性が優先される。
例をあげる。

```
<def tag="Header" size="15">
  <Text texts="$Args{text}">
    <style><clone caller="$TStyle{normal}" fontsize="$Args{size}"/></style>
  </Text>
</def>
```

このように定義されたときに、

```
<Header text="はじめに"/>
```

を実行すると、結果として次のものが実行されることになる。

```
  <Text texts="はじめに">
    <style><clone caller="$TStyle{normal}" fontsize="15"/></style>
  </Text>

<Header text="はじめに" size="20"/>
```

とすると <def> での size="15" が上書きされて、次のものが実行されることになる。

```
  <Text texts="はじめに">
    <style><clone caller="$TStyle{normal}" fontsize="20"/></style>
  </Text>
```

<def> 自体は何も結果を返さない。
<def> の属性としては、次のものが指定できる。

tag
: 定義するタグ名を指定する

attributesname
: 実行時の属性(＝定義されたタグ名の要素の属性)をまとめてハッシュ参照とし、引数として %Args

で渡したいときにその名前を指定する。 <def> の内容では、attributes という属性にその $Args{…} 変数を指定する。

contentsmode

実行時の内容（＝定義されたタグ名の要素の内容）をどのように扱うかを指定する。
指定無し：引数モードで処理されて、属性と合わせて引数となる
method：メソッドモードで処理して、その結果の配列参照を、contentsname で指定した名前の引数とする
text：method と同様だが、テキストを無視せず結果に含める
autonl：text と同様だが、テキスト内の改行を NewLine に変換する
raw：処理せずに XML::Parser が返した配列参照（ただし属性のハッシュ参照は取り除かれている）のまま、その時点のローカルモードのレベル数とともに、contentsname で指定した名前の引数とする。この値は、 <do> の contents 属性の値として与えれば処理して結果を得ることができる。
raw 以外の場合は、 <def> の内容が処理される前に実行時の内容が処理されて、 <def> の内容が処理される時の引数となる。raw の場合は、 <def> の内容が処理される時に実行時の内容は処理されておらず、 <def> の内容の中で自分で <do contents=...> で処理することになる。

contentsname

contentsmode を指定したときに、内容の処理結果が渡される引数の名前を指定する。省略すると 'contents' となる。

contentsmode を指定した例をあげる。

```
<def tag="Style" contentsname="text" contentsmode="autonl" attributesname="attr">
  <Text>
    <texts>$Args{text}</texts>
    <style><TStyle attributes="$Args{attr}"/></style>
  </Text>
</def>
```

このように定義されたときに、

```
<Style withline="1"> 下線 <Style italic="1"> 下線で斜体 </Style></Style>
```

と実行されると、まず外側の <Style> が置き換えられて、

```
<Text>
  <texts> 下線 <Style italic="1"> 下線で斜体 </Style></texts>
  <style><TStyle withline="1"/></style>
</Text>
```

となり、さらに内側の <Style> が置き換えられて、

```
<Text>
  <texts> 下線 <Text>
    <texts> 下線で斜体 </texts>
    <style><TStyle italic="1"/></style>
  </Text></texts>
  <style><TStyle withline="1"/></style>
</Text>
```

となる。

■ローカルモード

ローカルモードでタグが実行されるのは、マクロ定義されたタグ、<do> に local="1" が指定されたとき、<do> に追加の属性があるとき、の三つの場合である。
ローカルモードでタグが実行されるときには、開始時に属性が %Args に追加セットされ、終了時には %Args は開始時の状態に戻る。また、ローカルモード内で <local> で局所化した変数の値も、終了時に元に戻る。
例えば次のように実行すると、

```
<do A="1" B="2">
  <eval>print "Outer: A=$Args{A} B=$Args{B}\n"</eval>
  <do A="3">
    <eval>print "Inner: A=$Args{A} B=$Args{B}\n"</eval>
  </do>
  <eval>print "Outer: A=$Args{A} B=$Args{B}\n"</eval>
</do>
```

次のように表示される。

```
Outer: A=1 B=2
Inner: A=3 B=2
Outer: A=1 B=2
```

<do contents="…"> では、contents で指定された実行内容が書かれた場所での %Args がベースとなることに注意。例えば次のように実行すると、

```
<def tag="T" contentsmode="raw">
  <eval>print "In  T: A=$Args{A} B=$Args{B}\n"</eval>
  <do contents="$Args{contents}" B="4"/>
  <eval>print "In  T: A=$Args{A} B=$Args{B}\n"</eval>
</def>
<do A="1" B="2">
  <eval>print "Outer: A=$Args{A} B=$Args{B}\n"</eval>
  <T A="3">
    <eval>print "Inner: A=$Args{A} B=$Args{B}\n"</eval>
  </T>
  <eval>print "Outer: A=$Args{A} B=$Args{B}\n"</eval>
</do>
```

<T> の中での「<do contents="$Args{contents}" B="4"/>」を実行するとき、$Args{contents} である「<eval>print "Inner: A=$Args{A} B=$Args{B}\n"</eval>」は、<T> の中の %Args でなく外の %Args をベースにして実行されることになる。この結果、次のように表示される。

```
Outer: A=1 B=2
In  T: A=3 B=2
Inner: A=1 B=4
In  T: A=3 B=2
Outer: A=1 B=2
```

■注意

XPDFJ はαバージョンの状態にあり、その仕様は変わる可能性が大いにある。そのつもりで使っていただきたい。

■サンプル

マクロ定義のサンプルとして、stddefs.inc、stdfontsH.inc、stdfontsV.inc が添付されている。stddefs.inc 自体の中に簡単な説明がある。これらの使い方のサンプルとしては、of2002.xp を参照。
これらのファイルの内容は今後大きく変わる可能性があることに留意して使っていただきたい。

AUTHOR

中島 靖 nakajima@netstock.co.jp http://hp1.jonex.ne.jp/~nakajima.yasushi/

SEE ALSO

PDFJ