インターフェイスの街角 (27)– SWIG

増井 俊之

Perl や Tcl などのスクリプト言語は手軽に利用できる半面、言語に用意されていない機能は使いづらいという問題がありました。たとえば、Perl で高速な検索システムを構築したり、3 次元グラフィックスの表示や MP3 ファイルの再生をおこなうのは容易なことではなく、Tcl で UNIX のシステムに深く依存する処理を呼び出すのも簡単ではありません。最近のスクリプト言語の多くは、外部ライブラリをリンクする機能を備えているため、C のライブラリをスクリプト言語から使うこともできます。しかし、そのためのインターフェイス仕様は簡単とはいえず、しかも言語ごとに異なるので、スクリプト言語に C ライブラリをリンクして使う方法はあまり普及していないのが実情です。

このような問題を解決するために、SWIG というツールが作られています。SWIG は、C のライブラリをスクリプト言語から簡単に呼び出せる形式に変換するツールで、現時点では Tcl、Perl、Python に対応しています。

■ Ruby も対応しているようですが

SWIG を使えば、C で実装された特殊なライブラリも、Perl などから呼び出して使えるようになります。今回は、SWIG とその応用について紹介します。

複数の言語の組合せ

世の中に数多くのプログラミング言語やツールがあるのは、それぞれの言語に得手、不得手があるからでしょう。たとえば、テキストのパターンマッチ処理には Perl が向いていますし、ボタンやスライダなどの GUI をもつ簡単なアプリケーションであれば Tcl/Tk が便利です。高速な数値計算なら、C や Fortran を使うのがよいでしょう。このように仕事に適した言語がはっきりしている場合はいいのですが、

- ボタンやスライダで 3 次元グラフィックスを制御したい
- 曲名を曖昧検索して MP3 ファイルを再生したい

といったときにはどうすればよいのでしょうか。

前者の場合は、1999 年 8 月号で紹介したようなツールキットを使えば、GUI 部品で OpenGL 描画を制御することはできます。しかし、肝腎の描画処理よりも GUI のためのプログラミングに手間がかかりそうです。後者については、検索や画面表示のようなインターフェイス部分はスクリプト言語で十分であっても、再生部は C で書かねばならないので、けっきょくはすべてを C で実装することになるのではないでしょうか。

このように、現在のプログラミング言語では状況に応じた言語の使い分けが難しいので、1 つの言語ですべての処理をおこなうことが多いようです。たとえば、いったん Java を使うと決めたら、通信も画面表示もファイルアクセスもすべて Java で記述しなければならなくなることが多いように思われます。

状況に合わせて異なる言語を好きなように組み合わせて使えれば、システム開発はかなり楽になるでしょう。

- 数値計算は Fortran
- 数式の計算は Mathematica
- グラフィカルな配置などは視覚言語
- 3 次元表示は OpenGL
- 音楽は MAX
- システム関数は C
- パターンマッチやテキスト処理は Perl
- 簡単な GUI は Tcl/Tk

というふうに各種の言語を組み合わせて利用できれば、プログラムもずいぶん作成しやすくなるのではないでしょうか。

スクリプト言語と C 言語の組合せ

SWIG (Simplified Wrapper and Interface Generator) は、University of Chicago の Dave Beazley 氏が開発したシステムで、Tcl や Perl、Python などのスクリプト言語から C のライブラリを簡単に呼び出せるようにするツールです。最近のスクリプト言語は、動的ライブラリを使ったり、あるいは実行オブジェクトにライブラリを静的にリンクして C ライブラリを呼び出せるようにする仕組みをサポートしています。しかし、各言語の外部ライブラリ呼出し仕様に従ってユーザーのライブラリを実装する必要があり、外部ライブラリ呼出しの仕組みも言語ごとに異なります。したがって、現実にはこのような機能はあまり利用されていません。ところが SWIG を使えば、Perl や Python などから普通の形式の C 関数をいとも簡単に呼び出せるようになります。

Beazley 氏によれば、SWIG には次のような利点があるそうです。

- 既存の C プログラムに強力なインターフェイスを付加できる。
- ラピッド・プロトタイピングに便利である。
- 対話的デバッグが可能。
- Tk などを用いて GUI が追加できる。
- C ライブラリのテストが容易になる。
- スクリプト言語用の高性能な C モジュールが簡単に作れる。
- C のプログラミングが楽しくなる。
- 人を驚かすことができる。

同じく Beazley 氏によれば、そもそも C によるプログラミングは次の部分の組合せで成り立っています。

1. いろいろな関数と変数の集合
2. プログラムを開始するための main()
3. プログラムを適切に使えるようにするための各種の工夫

現実には、本質的な部分である 1 よりも、2 や 3 ばかりが複雑になって扱いきれなくなることが多い。しかし、1 の部分だけを C で実装し、残りはスクリプト言語を使うことにすれば、プログラムは、

1. いろいろな関数と変数の集合
2. それを使うためのスクリプト言語

という構成になり、プログラムが格段に扱いやすくなる。これが Beazley 氏の主張です。

たとえば、プログラムの引数処理やインターフェイスを Tcl/Tk で作成し、本質的な部分だけを C のライブラリとして実装すれば、見栄えやオプションの対話的な変更、追加が容易になります。C で大きなライブラリを作ったとき、すぐにそれが Perl でも Tcl でも使えれば嬉しいですし、ライブラリのテストを対話的におこなえるのは便利です。そのようなライブラリをコマンドとして動かす場合も、C で main() を書かずに引数処理部だけを Perl で実装する、といった発想はおもしろいかもしれません。

ユーザー・インターフェイスの分野では、**アプリケーション部とインターフェイス部をいかに分離して実装するか**という研究がかなり以前からおこなわれています。これがうまく実現できれば、以下のようなメリットがあると考えられるからです。

- インターフェイス・プログラムの再利用ができる
  異なるアプリケーションに対し、同じインターフェイス・モジュールが使えるようになります。
- 1 つのアプリケーションに異なるインターフェイスが使える
  インターフェイス部だけを交換することによって、同じプログラムがウィンドウ・システム上でも文字端末上でも使えるようになります。さらに、ユーザーの習熟度に応じて異なるインターフェイスを適用するのも簡単です。また、インターフェイスのカスタマイズも容易になります。
- テストモジュールやデモモジュールを簡単に切り替えられる
  ユーザーが操作する部分だけを手軽に別のものに置き換えられるのなら、テストやデモなどを苦労せずに実行できます。

残念ながら、C のような普通の手続き型言語では、プログラムのインターフェイス部とアプリケーション部を分離

するのは困難です。逐次実行文を処理内容に応じて分割するのは容易ではなく、さらにインターフェイスの見栄えがアプリケーション内部の状況に依存する場合が多いため[1]、両者をうまく分離するのは本質的に難しいのです。

SWIG は、このような問題を解決してくれるわけではありません。しかし、完全に一枚岩のアプリケーション構造にくらべると、複数の言語間で機能を分担できる余地があることは大きなメリットです[2]。

SWIG は、Windows 95/98 にも対応しているので、Windows 上の Perl や Tcl などからも同じように C の関数を呼び出すことができます。Windows 専用のツールキットでプログラミングをするのではなく、UNIX と Windows で同じプログラムが動くのは嬉しいところです。

SWIG の使い方

SWIG の最新版は下記の Web ページで入手できます。

●http://www.swig.org/

SWIG は多彩な機能をもっていますが、以下では基本的な使い方に絞って紹介します。

ライブラリの用意

まず、スクリプト言語から呼び出したい C のライブラリ関数を用意します。今回は、1998 年 2 月号などで紹介した曖昧検索ライブラリをスクリプト言語から呼び出すことにしましょう。

Perl や Tcl には、正規表現によるパターンマッチの機能があらかじめ用意されています。しかし、1 文字誤りや 2 文字誤りを許すといった、〝曖昧度〟を指定して検索する機能はありません。C 言語であれば、このような機能は簡単に実現できますが、スクリプト言語の場合には速度の面で問題があります。そこで、曖昧検索用の関数を SWIG を用いて Perl や Tcl の追加ライブラリ関数として登録してみることにしました。

図 1-a が曖昧検索ライブラリの本体、図 1-b がヘッダファイルです。makepat() で検索パターンと曖昧度を指定して初期化し、match() でマッチングをおこないます。この 2 つの関数をスクリプト言語から呼び出せるようにします。

SWIG では、関数のインターフェイス仕様を定義するため、プログラム本体とは別に図 1-c のような定義ファイルを用いてスクリプト言語から呼び出す関数やライブラリ・モジュールの名前を指定します。

ご覧のように、この定義ファイルはごく簡単なものですが、スクリプト言語からライブラリを呼ぶために用意するのはこれだけです。

Tcl からの使用

Tcl 8.0 から makepat() を呼べるようにするには、以下の手順で共有ライブラリを作成します。

まず、**swig** コマンドで asearch.i から〝Wrapper ファイル〟を作ります。

```
unix% swig -tcl8 asearch.i
Making wrappers for Tcl 8.x
unix%
```

これで、asearch_wrap.c が生成されます（図 2）。ここでいう Wrapper とは、Tcl から makepat() を呼び出せるようにするための定義をまとめたものです。

ライブラリ本体と Wrapper を以下のようにしてコンパイルすると、Tcl からロードできる共有ライブラリ asearch.so が生成されます。

```
unix% gcc -c -fpic asearch.c asearch_wrap.c
unix% gcc -shared asearch.o asearch_wrap.o \
-o asearch.so
vunix%
```

この asearch.so を Tcl から読み込めば、曖昧検索ライブラリが呼び出せるようになります（図 3）。

このように asearch.i を新たに定義するだけで、Tcl から曖昧検索ライブラリが呼び出せるようになります。

曖昧検索を用いた辞書検索システム

曖昧検索モジュールが Tcl/Tk で使えるようになったところで、これを利用して辞書の曖昧検索アプリケーショ

1 アプリケーションの状態に応じてメニュー表示を変化させる（たとえば、灰色にして選択不能にする）ような場合、アプリケーション部のもつデータはインターフェイス部と共有されなければなりません。

2 いろいろなシステムをとりまぜて使うには、なんらかの共有空間を介して疎結合で通信させるのがもっとも効果的な方法だと思います。とくに、これからのインターフェイスはマウスや画面だけを扱えばよいわけではなく、音声や実世界の機器など、多種多様な装置への対応が求められるでしょう。そのような場合、1 つの言語ですべての装置の面倒をみるのは事実上不可能ですから、個々の装置とそれに対応する言語をいかにうまく組み合わせて使えるかが重要なポイントとなってきます。

図 1　曖昧検索ライブラリ関連プログラム

(a) asearch.c（ライブラリ本体）

```
#include "asearch.h"

#define INITPAT 0x40000000

static int mismatch;
static unsigned int epsilon, acceptpat;
static unsigned int shiftpat[MAXCHARCODE];

void
makepat(char *pat, int m)
{
  int i;
  unsigned int mask = INITPAT;
  unsigned char *p = (unsigned char*)pat;

  mismatch = m;
  epsilon = 0;

  for(i=0;i<MAXCHARCODE;i++) shiftpat[i] = 0;
  for(;*p;p++){
    if(*p == ' '){ // ワイルド カード 文字
      epsilon |= mask;
    }
    else {
      shiftpat[*p] |= mask;
      if(isupper(*p)){
        shiftpat[tolower(*p)] |= mask;
      }
      else if(islower(*p)){
        shiftpat[toupper(*p)] |= mask;
      }
      mask >>= 1;
    }
  }
  acceptpat = mask;
}

int match(register char *text)
{
  register unsigned int i0 = INITPAT, i1=0, i2=0, i3=0;
  register unsigned int mask;
  register unsigned int e = epsilon;

  for(;*text;text++){
    mask = shiftpat[*(unsigned char*)text];
    i3 = (i3 & e) | ((i3 & mask) >> 1) | (i2 >> 1) | i2;
    i2 = (i2 & e) | ((i2 & mask) >> 1) | (i1 >> 1) | i1;
    i1 = (i1 & e) | ((i1 & mask) >> 1) | (i0 >> 1) | i0;
    i0 = (i0 & e) | ((i0 & mask) >> 1);
    i1 |= (i0 >> 1);
    i2 |= (i1 >> 1);
    i3 |= (i2 >> 1);
  }
  switch(mismatch){
    case 0: return (i0 & acceptpat);
    case 1: return (i1 & acceptpat);
    case 2: return (i2 & acceptpat);
    case 3: return (i3 & acceptpat);
    default: return 0;
  }
}
```

(b) asearch.h（曖昧検索ライブラリヘッダ）

```
#define MAXCHARCODE 0x100
#define MAXMISMATCH 3

void makepat(char *pat, int m);
int match(char *text);
```

(c) asearch.i（ライブラリ定義ファイル）

```
%module asearch

void makepat(char *pat, int m);
int match(char *text);
```

図 2　asearch_wrap.c（SWIG によって生成される Wrapper）

```
    ......
#   define SWIGEXPORT(a,b) a b
    ......
static int _wrap_makepat(ClientData clientData,
   Tcl_Interp *interp, int objc, Tcl_Obj *CONST objv[]) {

    char * _arg0;
    int  _arg1;
    Tcl_Obj * tcl_result;
    int templength;
    int tempint;

    clientData = clientData; objv = objv;
    tcl_result = Tcl_GetObjResult(interp);
    if ((objc < 3) || (objc > 3)) {
        Tcl_SetStringObj(tcl_result,"Wrong # args. makepat pat m ",-1);
        return TCL_ERROR;
    }
    if ((_arg0 = Tcl_GetStringFromObj(objv[1], &templength)) == NULL) return TCL_ERROR;
    if (Tcl_GetIntFromObj(interp,objv[2],&tempint) == TCL_ERROR) return TCL_ERROR;
    _arg1 = (int ) tempint;
    makepat(_arg0,_arg1);
    tcl_result = Tcl_GetObjResult(interp);
    return TCL_OK;
}

static int _wrap_match(ClientData clientData, Tcl_Interp *interp,
                       int objc, Tcl_Obj *CONST objv[]) {
    ......

}

SWIGEXPORT(int,Asearch_Init)(Tcl_Interp *interp) {
 if (interp == 0)
 return TCL_ERROR;
 SWIG_RegisterType();
 Tcl_CreateObjCommand(interp, SWIG_prefix "makepat", _wrap_makepat, (ClientData) NULL,
                              (Tcl_CmdDeleteProc *) NULL);
 Tcl_CreateObjCommand(interp, SWIG_prefix "match", _wrap_match, (ClientData) NULL,
                              (Tcl_CmdDeleteProc *) NULL);
    ......
```

図 3　曖昧検索ライブラリの呼出し

```
unix% tclsh8.0jp
% makepat " abc " 1              ←共有ライブラリのロード前
invalid command name "makepat"  ←makepatは未定義
% load ./asearch.so             ←共有ライブラリをロード
% makepat " abc " 1              ←makepatで検索パターンと曖昧度を指定
% match "abcdefg"
134217728                       ←パターンにマッチ
% match "abkdefg"
134217728                       ←パターンにマッチ（1文字誤り）
% match "akkdefg"
0                               ←2文字以上の誤りはマッチング失敗
%
```

図 4　asearch.tcl（曖昧検索辞書アプリケーション）

```
load "asearch.dll"

set dic "/usr/dict/words"
set f [open $dic]

set line 0                          # 辞書の読出し
while {![eof $f]} {
  set d($line) [gets $f]
  incr line
}
close $f

text .pat -height 1 -width 20       # パターン入力枠
text .result -height 10 -width 20   # 結果表示枠
pack .pat .result

foreach i [bind Text] {             # 既存のバインディングを消去
  bind .pat $i [bind Text $i]
  bind Text $i ""
}

proc searchKey { w a } {
  tkTextInsert $w $a
  search [.pat get 1.0 1.end]
}

for { set c 97 } { $c < 122 } {incr c} {
  bind .pat [format "<Key-%c>" $c] { searchKey %W %A }
}

proc bs { w } {
  tkTextSetCursor $w insert-1c
  $w delete insert

  set pat [.pat get 1.0 1.end]
  set len [string length $pat]
  if { $len > 0} {
    search [string range $pat 0 [expr $len - 1]]
  } { search "" }
}

bind .pat <BackSpace> "bs %W"
bind .pat <Delete> "bs %W"

proc search { pat } {  # 各単語とpatとのマッチングを調べる
  global d line
  .result delete 1.0 end
  set n 0
  # 何か単語がみつかるまで曖昧度を増やしていく
  for {set a 0 } { $a < 4 && $n == 0} { incr a } {
    makepat "$pat " $a
    for { set i 0 } { $i < $line && $n < 10 } { incr i } {
      if {[ match $d($i) ]} {
        .result insert end "$d($i)\n"
        incr n
      }
    }
  }
}
```

図 5　ma で検索

図 6　masui で検索

ンを作ってみましょう。

図 4 は、前述の asearch モジュールを利用した辞書検索用のアプリケーション・プログラムです。

英単語の辞書に対し、検索パターンとして `ma´を指定して検索すると、`ma´で始まる単語が一覧表示されます（図 5）。同様に `masui´で検索すると、該当する英単語がないために `massif´などの綴りが似ている単語が表示されます（図 6）。

Perl からの使用

Perl での使い方も、Tcl の場合とほとんど同じです。まず、Wrapper (asearch_wrap.c) を生成します。

```
unix% swig -perl5 asearch.i
Making wrappers for Perl 5
unix%
```

続いて、以下のようにして共有ライブラリを生成します（誌面の都合上、⇒ で折り返しています）。コンパイル時には Perl のインクルード・ファイルが必要になります（以下に示すのは alpha-linux での例です）。

```
unix% set perldir=/usr/local/lib/perl5/⇒
5.00503/alpha-linux/CORE
unix% gcc -I$perldir -c -fpic asearch.c \
asearch_wrap.c
unix% gcc -shared asearch.o asearch_wrap.o \
-o asearch.so
unix%
```

生成した共有ライブラリ asearch.so は、以下のようにモジュールとして使うことができます。

```
unix% perl
use asearch;
asearch::makepat(" abc ",1);
print asearch::match("abcdefg"),"\n";
print asearch::match("abkdefg"),"\n";
print asearch::match("akkdefg"),"\n";
Ctrl-D
134217728
134217728
0
unix%
```

このようなライブラリがあれば、曖昧検索用の grep なども簡単に作れます。

```
#!/usr/local/bin/perl
use asearch;
$pat = shift;
asearch::makepat(" $pat ",1);
while(<>){
  print if asearch::match($_);
}
```

Windows での使用

さきほども述べたように、SWIG は Windows 上の Tcl や Perl、Python にも対応しています。Windows の場合は DLL が生成され、Windows 版の Perl や Tcl から UNIX 版と同様の方法でライブラリを呼び出すことができます。

SWIG の活用法

SWIG は、スクリプト言語の外部インターフェイスを使いやすくするだけのものともいえます。しかし、複数の言語の得意な部分を簡単に組み合わせることができるという点で、かなりのインパクトがあるように思えます。

私の従来の印象では、

表 1　UNIX コマンドへの応用

モジュール	対応する UIX コマンド
コード変換モジュール	nkf
かな漢字モジュール	jserver
漢字かなモジュール	kakasi
圧縮モジュール	gzip
暗号化モジュール	key
MIME デコード・モジュール	nkf
予測入力モジュール	(POBox)
曖昧検索モジュール	(asearch)
OpenGL モジュール	
辞書モジュール	
MP3 モジュール	

```
main(int argc, char **argv)
```

と書くのが男らしい (?) 本格的なプログラムで、スクリプト言語はいわばおもちゃのようなもの、遅いけれども一応はこういうこともできるといった 2 級市民的感覚が拭えませんでした。しかし、SWIG によって C による〝本格的なライブラリ関数〟とスクリプト言語による使いやすい〝インターフェイスの提供〟が融合できるのであれば、認識をあらためなくてはなりません。本質的なライブラリだけを SWIG モジュールにしておき、状況に応じて各種のスクリプト言語と組み合わせて利用する方法は、Beazley 氏が言うようになかなか魅力的です。

UNIX はもともと、〝(grep や sort などの) 特徴的な仕事をする小さなコマンドをシェル上で組み合わせて使える〟ことが利点でした。しかし、これからは〝特徴的な仕事をするライブラリをスクリプト言語上で組み合わせて使える〟というところに重心が移っていくかもしれません。

表 1 に示したように UNIX コマンドに対応した機能を SWIG モジュールとして用意しておけば、活用の幅がさらにひろがるのではないでしょうか。

SWIG の限界

以上に紹介したように SWIG はたいへん便利な機能をもっていますが、万能というわけではなく、以下に記すような限界があります。

まず、各種のスクリプト言語の外部インターフェイス仕様に準拠する必要があるので、

- つねにスクリプト言語の最新版に対応しているわけではない

という問題があります。事実、いくつかの機能は Tcl 7 でしか使えず、Tcl 8 ではまだ利用できないようです。

当然のことながら、

- 言語ごとに SWIG コンバータを用意しなければならない

のも面倒です。このほかに、

- スクリプト言語の機能をすべて使えるわけではない

という問題もあります。スクリプト言語で特殊なデータ構造が扱えたとしても、C のライブラリでそのすべてに対応するのはほぼ不可能です。さらに、

- 言語の関係が対等ではない

ことも問題になりそうです。SWIG では、スクリプト言語側をメインルーチンとして実現しなければなりません。したがって、GLUT や C のツールキットなど、メインルーチンが C のライブラリ内にあるものは扱いにくくなります。

おわりに

今日の有力なプログラミング言語では、sin() や exp() といった数値処理関数は標準的に用意されています。しかし、通信や圧縮、暗号化、コード変換などの機能はあくまでオマケとして扱われています。このように比較的新しくてよく使われる機能は、とりあえず SWIG を使ってライブラリにしておき、プログラミング言語に標準的に実装されるまで待つのがよいかもしれません。

SWIG によって、ツール・プログラミングの常識が変わることを期待したいと思います。

（ますい・としゆき　ソニー CSL）