プログラム実行環境としてのEmacs

Emacs as Execution Environment of Programs

伊藤 誠

1 はじめに

コンピュータを利用する際、さまざまな用途(目的)に応じたアプリケーションを用いることが多い。しかし個人差はあるであろうが、長い間業務としてコンピュータを使い続けるとその利用形態は定形化(ほぼ同じアプリケーションを利用)してくるのではないだろうか。

プログラミングを生業とする人にとって、常にエディタは欠かすことのできないものであり、日々利用するアプリケーションである。したがってエディタの中でコンピュータ利用に関する全てのことを完結して行うことができればそれに勝るものはないであろう。

数あるエディタのなかでEmacsは従前より「Emacsは環境である」といわれてきた。エディタ本来の役割であるテキスト(プログラム)入力にとどまらず、Emacsの中で行うことの出きることは数多くある。例えば

- テキスト(プログラム)入力
- 電子メール(添付ファイルの表示、URLの表示)
- DTP(LaTeX)
- ネット検索(ブラウザ連動)・辞書検索
- UNIXのシェルとのインターフェース(ターミナル)
- ミュージックプレイヤー・ビデオプレイヤーとしてのインターフェース
- 数式処理(Maxima)・グラフ処理(gnuplot)・データベース処理(MySQL)のインターフェース
- クラウドサービスとの連携(Evernote、gmail、googleカレンダー)
- 電卓
- 簡易表計算
- スケジューラー
- ディレクトリエディタ(エクスプローラ)
- 画像ビューアー
- 各種入力支援
- プログラム実行支援

などがあげられる。筆者も全ての機能を利用しているわけではないが、例えば電子メールは日々利用するものであり、使い慣れたエディタと同様の操作方法 (カーソル移動、検索、カット (コピー)・アンド・ペースト等) でエディタから離れることなく、電子メールの機能を利用できるのは代えがたいものがある。もちろん適材適所で必要に応じたアプリケーションを利用する場合もある。

上記の機能はほぼ全て Emacs Lisp(以後 elisp) で実現されている。elisp は実効速度が遅いと言われているが、コンピュータのハードウェアの進歩に伴い今後もその機能の拡張は続くと思われる。

2　入力支援

2.1　Emacs のメジャーモード

「はじめに」で列挙した Emacs の機能のうち、プログラミングに関しては入力支援は重要であろう。Emacs には各種プログラミング言語に応じたメジャーモードがあり、キーワードの色分など視覚的支援をしてくれるのはもちろんであるが、

- 括弧の対等の自動入力
- キーワードのコマンドでの入力
- キーワードや既入力文字の自動補完
- 自動インデント
- マニュアル参照 (一部の言語)
- 埋め込み型言語に対するマルチモード

などの支援を行ってくれる。

2.2　マルチモード

埋め込み型言語 (例えば HTML 文書の中に埋め込まれた PHP プログラムや JSP(JavaServer Pages) の場合は、HTML を編集している部分と PHP や Java などのプログラムを編集している部分では Emacs のメジャーモードは異なるものでなければ色分けなどが適切に行われない。Emacs には multi-mode.el という elisp があるので例えば

```
(autoload 'multi-mode "multi-mode" nil t)

(defun html-php-mode ()
  (multi-mode 1
     'html-mode
     '("<?php" php-mode)
     '("?>" html-mode)))
```

```
(defun jsp-mode ()
  (multi-mode 1
     'html-mode
     '("<%" java-mode)
     '("%>" html-mode)))
```

とすることによって、PHP プログラムの部分 (<?php と ?> で囲まれた部分) にカーソルがあればメジャーモードが PHP モードになり、それ以外では HTML モードになる。JSP に関しても同様である。

現在多数の Emacs 関連の書籍が出版されているが、その多くは Emacs の基本的使用方法の解説で終わっていたり、入力支援の設定方法に終始している。しかしプログラムを学習、あるいは作成するには実際にプログラムを実行してみる必要がある。にもかかわらず相変わらず、プログラムの入力は「エディタ」で、実行は「シェル (ターミナル)」で行う場合が多いようである。プログラムの実行支援に関する記事はあまり見受けられない。

そこで以下の節では、Emacs 内でプログラム実行を行う「実行環境としての Emacs」についてまとめてみることにする。

3　実行支援

3.1　LaTeX

TeX による文書整形をプログラミングと言ってよいかどうかには様々な意見があると思われるが、HTML 文書も含め、ここでは一種のプログラミング言語とみなすこととする。

LaTeX 関係の入力・実行支援として AUCTeX[1] や YaTeX(野鳥)[2] などがある。これらは LaTeX モードに付随して利用することができ、あらかじめ様々な機能がコマンド (キー操作で利用可能) として提供されている。例えば AUCTeX の入力支援として、様々な LaTeX 環境の入力コマンド `LaTeX-environment` がある。これは `C-c C-e` に割り当てられていて、コマンドを実行するとミニバッファに

```
Environment type: (default xxxx)
```

と表示されるので入力したい環境名を指定すればよい。

また TeX 関連の外部処理コマンドを実行する `TeX-command-master` は `C-c C-c` に割り当てられているので、LaTeX で処理したければ `pLaTeX` とし (プログラムのコンパイルに相当)、`xdvi` コマンドでプレビュー (プログラムの実行に相当) したければ `View` を選択すればよい。

ここまでは AUCTeX 付属の機能である。ここではもう一歩進めて現在編集している部分 (カーソルのある部分) をプレビューしたり、プレビュー画面の表示部分のソースへ移動する方法についてまとめる。まず必要な elisp を読み込む

```
(require 'server)
(unless (server-running-p)
  (server-start))
(require 'xdvi-search)
```

LaTeX で処理後 (platex コマンドに-src オプションをつけて処理する必要がある)、現在 Emacs で編集している部分 (カーソルのある部分) をプレビューするには xdvi-jump-to-line コマンドを実行する[1]。逆に現在プレビューしている部分の Emacs のソースへ移動するには .Xresources ファイルに

```
XDvi*editor: emacsclient --no-wait +%l %f
```

と記述しておき、プレビュー画面をコントロールキーを押しながらマウスで左クリックすればよい。これで Emacs の編集部分と xdvi のプレビュー画面の相互移動が可能となる[2]。WYSIWYG(What You See Is What You Get) ではない LaTeX での文書処理の実行環境としての一助となるであろう。

3.2　コンパイラ系言語

Emacs にはプログラムをコンパイルするコマンドとして compile がある。しかし実際に実行されるコマンドは make -k であり、これは複数のソースファイルをまとめて Makefile で処理することが前提となっている。このままでは今編集中のファイル (例えば C 言語のプログラム) を単独で cc コマンドで処理することはできない。そこで次のようなコマンドを作成しておく。

```
(defun invoke-cc-command ()
  (interactive)
  (setq command-line (concat "cc " (file-name-nondirectory
                                    (buffer-file-name))  "\n"))
  (compile command-line))
```

buffer-file-name コマンドで現在編集中のファイルのフルパス名 (絶対パス名) がえられるので、file-name-nondirectory でファイル名だけ取り出し cc コマンドに渡すコマンドラインを変数に代入しておき、compile コマンドを実行する。別バッファがポップアップされコンパイル結果が表示される。エラーがあれば C-c ‘でエラー箇所に移動できる。C 言語に限らずコマンドの処理結果は自動的に別バッファにポップアップ表示されるが[3]、この時ポップアップ表示された別バッファを適切に表示・削除するには popwin.el[3] を利用するとよいであろう。

コンパイルしたファイルを実行するには (上記の elisp では実行ファイルは a.out となる)shell-command に a.out &を、あるいは async-shell-command コマンドに a.out を渡せばよい[4]。Emacs のコマンドとして登録しておけばよいであろう。

[1]適当なキーに割り当てておけばよいであろう。
[2]ただしカーソル位置やマウスでクリックした位置にピンポイントで移動するわけではない。
[3]LaTeX で処理する場合もエラーがあれば別バッファにポップアップ表示される。
[4]実行時引数が必要な場合は別途設定が必要である。

次に Java プログラムの実行を考える。Java は C 言語と異なり、ファイルの階層構造で package を管理するようになっている。この階層構造に合わせてコンパイル・実行しなければならない。つまり必ずしも現在編集中のファイルがあるディレクトリでコンパイル・実行すればよいわけではない[5]。編集中のファイルから適切に package 名を取得し、ディレクトリの階層構造を移動してコンパイル・実行しなければならない。例えばファイル名 Sample.java の 1 行目に

```
package aaa.bbb.ccc;
```

という package 文があればファイルが存在するディレクトリから移動し、次のようにコンパイルしなければならない、

```
cd ../../..; javac aaa/bbb/ccc/Sample.java
```

また実行するには、同様にディレクトリを移動してから

```
cd ../../..; java aaa.bbb.ccc.Sample
```

としなければならない。

この処理を Emacs 内で行うことを考える。ここでは簡単のため上記の例のように、package 文が宣言されている場合はそれがファイルの 1 行目の最初から始まっているものとする。まず package 文が宣言されているかどうかは、

```
(defun package-name-p ()
  (save-excursion
    (goto-char 1)
    (if (string= "package"  (buffer-substring 1 8))
       t
      nil)))
```

とすれば package 文の有無で真偽値が返る。次に pacage 名を取得するには

```
(defun get-package-name ()
  (save-excursion
    (goto-char 1)
    (if (string= "package"  (buffer-substring 1 8))
        (buffer-substring 9 (- (point-at-eol) 1)))))
```

とする。package 文がある場合、package 名にしたがってコンパイルするには次のように package 名からから上位の階層ディレクトリへ移動するパス名を `cd-path-name` に代入し、移動した後でコンパイルするためのソースファイルのパス名を `src-path-name` に代入する。これらを結合してコマンドラインを作成すればよい。当然のことであるがこれらの変数の生成には正規表現[6]を用いた文字列の置換を行う必要がある。

[5] もちろん package 文がない場合は前述の C 言語の場合と同じである。

[6] Emacs の正規表現は通常の正規表現 (例えば PCRE(Perl-Compatible Regular Expressions)) とは異なるので注意が必要である。

```
(defun invoke-javac-command ()
  (interactive)
  (cond ((package-name-p)
    (setq cd-path-name
        (replace-regexp-in-string "[^/]+" ".."
        (replace-regexp-in-string "\\." "/" (get-package-name))))
         (setq src-path-name
           (concat (replace-regexp-in-string "\\." "/"
             (get-package-name))  "/"
                   (file-name-nondirectory (buffer-file-name))))
         (setq command-line
               (concat "cd " cd-path-name "; javac " src-path-name))
         (compile command-line)
         )
        (t
         (setq src-path-name (file-name-nondirectory (buffer-file-name)))
         (setq command-line  (concat "javac " src-path-name))
         (compile command-line))))
```

また実行するにはコンパイルの場合と同様に移動するパス名を cd-path-name に、移動した後で実行するクラスファイルをのパス名を class-path-name 代入し、結合してコマンドラインを作成する。

```
(defun exec-java()
  (interactive)
  (cond ((package-name-p)
         (setq cd-path-name
            (replace-regexp-in-string "[^/]+" ".."
            (replace-regexp-in-string "\\." "/" (get-package-name))))
          (setq class-path-name
            (concat (get-package-name) "."
         (file-name-nondirectory
               (file-name-sans-extension (buffer-file-name)))))
          (setq command-line
               (concat "cd " cd-path-name ";java " class-path-name "&"))
         (shell-command command-line))
        (t
         (setq command-line (concat "java "
         (file-name-nondirectory
               (file-name-sans-extension (buffer-file-name))) "&"))
         (shell-command command-line))))
```

これらも適当なキーに割り当てておけばよいであろう。

この様な方法で、シェル (ターミナル) へ移動することなく、コンパイル・実行を Emacs 内で行うことができる。

3.3 インタプリタ系言語

インタプリタ系言語 (シェルスクリプト、Perl、Ruby、python 等) の実行は、コンパイラ系よりも簡単である。Emacs にはもともと executable-interpret コマンドがある。このコマンドに現在編集中のファイル名を渡せばよいので、

```
(defun exec-script ()
  (interactive)
   (executable-interpret (buffer-file-name)))
```

といったコマンドを作成しておけばよい。ただしファイルには実行属性が必要であるため、ファイル保存時に自動的に実行属性を与えるようにしておく。

```
(add-hook 'after-save-hook
          'executable-make-buffer-file-executable-if-script-p)
```

このコマンドはシバン (shebang(1 行目の#!)) があれば自動的にファイルに実行属性を与える。

3.4 Web 系

Web 系文書には単独の HTML 文書だけでなく、JavaScript、PHP、JSP といったプログラミング言語が埋め込まれているものがある。これら全ての場合に入力はエディタで、表示はブラウザ (プログラムの実行に相当) で行う。表示は編集中のファイルを「ファイル」として表示する場合と、Apache 等の Web サーバ経由で表示する場合がある。find-file-at-point コマンドはファイル名が URL 形式であれば自動的にブラウザに表示を行う。したがって現在編集中のファイルを「ファイル」としてブラウザに表示するには、ファイル名の先頭に file://を付加すればよいので、例えば

```
(defun invoke-browser-with-file ()
  (interactive)
  (save-some-buffers)
  (find-file-at-point (concat "file://" (buffer-file-name))))
```

などとし、Web サーバ経由で個人のファイル (~/public_html 内のファイル) を localhost に表示するには

```
(defun invoke-browser-on-localhost ()
  (interactive)
  (save-some-buffers)
  (find-file-at-point (concat "http://localhost/~mitoh/"
                (file-relative-name (buffer-file-name) "~/public_html"))))
```

などとすればよい。既定のブラウザ (Ubuntu の場合は firefox) に表示される。ブラウザを変更するには

```
(defvar *chrome-path* "/usr/bin/chromium-browser")
(setq browse-url-browser-function
      '(("."         . browse-url-generic)))
(setq browse-url-generic-program *chrome-path*)
```

とする。この場合ブラウザは chromium に変更される。

4　結語

プログラム開発においては、eclipse などの統合開発環境 (IDE:Integraed Development Environment) がある。多人数による大規模なアプリケーション開発によく用いられている。至れり尽くせりの機能を利用することができる。ただしこれらの IDE は基本的にプログラム開発に特化したものであり「はじめに」で述べたコンピュータ利用の「環境」とは言えない。Emacs は本来エディタでありながら、elisp による拡張性を備えているため様々な機能を実装し「環境」として進化してきた。コンピュータを利用する一つの統一された環境の中で、Emacs をプログラムの入力・実行環境として使えることは、プログラムの学習・作成の効率化に大きく寄与するであろう。

Emacs は GNU プロジェクトの代表的産物であり、自由に無料で利用することができる。また世界規模のコミュニティで日々機能が拡張されている。Emacs の今後の発展に期待したい [4][5][6]。

参考文献

[1] http://www.gnu.org/software/auctex/
[2] http://www.yatex.org/
[3] http://d.hatena.ne.jp/m2ym/20110120/1295524932
[4]「Emacs テクニックバイブル」 るびきち著 (技術評論社)
[5]「Emacs LISP テクニックバイブル」 るびきち著 (技術評論社)
[6]「Emacs 実践入門」 大竹智也著 (技術評論社)

Abstract

Emeacs is one of the editors well used on UNIX system. Originally an editor is used for the input of a text and a program. However, Emacs offers the environment where it was unified in the case of using a computer. This note discusses Emacs as not input support of a program but execution environment of a program.

Key words: Editor, Emacs, Programming, Execution Environment

キーワード:エディタ、Emacs、プログラム、実行環境