百万人のMutt
～Mutt活用講座～

滝澤隆史
taki@cyber.email.ne.jp

第10回　多言語メール環境

先月の記事の冒頭でも触れましたが、2001年12月に日本UNIXユーザ会主催の「The Linux/BSD Day」というイベントのMUAのセッションで、筆者はMuttについて紹介を行いました。その自由討論の中で「Mutt で日本語と韓国語の混在が可能か？」という質問がありました。そこで、筆者は「procmail やmaildrop などを使ってFrom ヘッダなどで振り分けを行い、folder-hookでcharsetを切り替える方法がある」、「UTF-8対応のxtermを使えばよい」と回答しました。

しかし、よくよく考えてみると、前者の方法だけでは実現できないことが分かります。実際にはUTF-8対応のターミナルエミュレータを使うことになります。そこで、今回は多言語メール環境をテーマとして、その実現方法を紹介します。

本題に入る前にMuttの最新状況をお知らせします。執筆時点での最新版は1.3.27iです。これは1.4の最終ベータ版（予定）です。先月号でもお知らせした通り、1.2.5iおよび1.3.24i以前にはセキュリティホールがありますので、そのバージョンを使用されている方は、1.2.5.1iおよび1.3.25i以降にバージョンアップしてください。また、Mutt の開発者メーリングリストでは、しばらく開発から遠ざかっていた作者のMichael Elkins氏が昨年末から活発に投稿するようになり、ヘッダをキャッシュするパッチ、SMTPの送信機能を追加するパッチなど、主要な機能を追加するパッチを投稿しています。これらのパッチはまだ実験的であり、次の開発版1.5系列には取り込まれていくでしょう。mutt-jメーリングリストでヘッダをキャッシュするパッチを当てた人がいて「とっても速くなって感激した」との報告があります*1。

それでは、本題に入りましょう。

必要なもの

Mutt の多言語環境は、扱う文字符号化方式を「UTF-8」にすることで実現します。そのため、UTF-8に関していくつか用意する必要があります。

まずは、文字コードに関する簡単な知識が必要です。ここで「UTF-8、ISO 10646、UCSって何？」という人がいましたら、本誌2001年11月号に掲載したこの連載のコラム「日本語の符号化文字集合と文字符号化方式」を読んでみてください*2。

続いて、Mutt が利用する次のライブラリとプログラムがUTF-8に対応している必要があります。

- C ライブラリ
- curses ライブラリ
- ターミナルエミュレータ
- エディタ
- auto_view で用いるプログラム

これ以外にも次のものが必要になる場合があります。

- UCS フォント
- UTF-8 対応の Input Method

ここで挙げたものについて具体的に紹介していきます。

●C ライブラリ

Mutt でUTF-8 を扱うためには、「UTF-8 localeに対応したC ライブラリ」が必要です。最近のLinuxのディストリビューションで使われている「glibc-2.2」は、UTF-8 locale に対応しています。しかしglibc-2.1のように、まともなUTF-8 Locale を持っていないC ライブラリを使っている場合は、libutf8（記事末のResources [2] を参照）をインストールする必要があります。libutf8のインストール方法については、コラム「libutf8 のインストール」をご覧ください。

なお、Mutt 内蔵のワイド文字関数を使う場合には、UTF-8 localeに対応したCライブラリやlibutf8がなくてもMutt でUTF-8 を扱うことができます。

●curses ライブラリ

Muttでよく使われるcursesライブラリには、S-Lang やncurses があります。

●S-Lang

S-Langライブラリ（執筆時点ではバージョン

Column　libutf8 のインストール

libutf8 は、UTF-8 locale に対応していないC ライブラリを使っている場合でもUTF-8 localeを使用できるように、Bruno Haible氏が作成したライブラリです。インストールは単に次のように行うだけです。

```
$ ./configure --prefix=/usr/local
$ make
# make install
```

glibc-2.1や2.0のように、ワイド文字関数を実装していて不具合のある環境では、実行例Aのような環境変数を設定します。実際にはGNU libiconvの設定と合わせて実行例Bのようになるでしょう。

これらのコマンドを.bashrcなどに追加しておいてください。　（滝澤隆史）

【実行例A】glibc-2.1、2.0 利用時の libutf8 用環境設定

```
$ LD_PRELOAD=/usr/local/lib/libutf8_plug.so
$ export LD_PRELOAD
```

【実行例B】glibc-2.1、2.0 利用時の libutf8 用環境設定（libiconv との組み合わせ）

```
$ LD_PRELOAD=/usr/local/lib/libiconv_plug.so:/usr/local/lib/libutf8_plug.so
$ export LD_PRELOAD
```

＊1　今月のコラム「Mutt も歩けば棒に当たる」を執筆された高橋氏の報告。　＊2　筆者のサイト（記事末の Resource【1】参照）で過去の掲載記事を公開しています。　＊3　Mutt に国際化機能を追加した人。

1.4.4、[3]）では、そのままUTF-8の文字を表示させると乱れた表示になります。そのため、Edmund G. Evans氏*3のUTF-8パッチ（[4]）を当てて修正する必要があります。

注意点としては、UTF-8パッチを当てたS-Langライブラリをインストールしたことで、既存のS-Langライブラリを利用しているプログラム（特にslang-jpを使っているプログラム）に影響が出るといけないので、実行例1のように、インストールせずにコンパイルだけしておくことです。そして、後ほどMuttをコンパイルするときには、実行例2のように静的にリンクするようにしてください。

【実行例1】S-Langライブラリのコンパイル

```
$ bzip2 -dc slang-1.4.4.tar.bz2 |tar xvf -
$ cd slang-1.4.4
$ patch -p0 < ../slang-1.4.4-ege2.diff
$ ./configure
$ make
$ make runtests
```

【実行例2】S-Langを静的にリンクしてMuttをコンパイルする

```
./configure --with-slang=../slang-1.4.4
```

Column

mlterm

mltermは、荒木賢氏が作成した多言語対応のターミナルエミュレータです。標準のインストールでUTF-8を扱うことが可能になっています。

mltermを使う最大の理由は、次に挙げる特徴に示すように、「既存の環境に手を加えないでもUTF-8を扱えること」です。UTF-8の環境が十分に整いきってはいない現状では非常に重宝します。

まず1つ目の特徴は、内部でUCSのマップデータを持っているため、UCSフォントがない場合でもUTF-8を扱えることです。そのため、新規にUCSフォントをインストールする必要はありません。

2つ目の特徴はUTF-8以外の文字の入力や張り付けを行った際に、UTF-8に変換してくれることです。そのため、XIMがUTF-8に対応していなくても文字の入力ができます。

また、背景を透過にしたり、画像を表示したりするターミナルエミュレータは他にもいくつかありますが、それに加えて、アンチエイリアス、マルチPTY、マルチXIMなど、ユニークな機能が実装されています。

mltermは日本人が開発しているだけあって、日本語の文書が付属しています。それを読めば、インストールや設定は簡単にできると思います。ここでは、UTF-8を扱うにあたって最低限必要な設定を紹介します。

まず、フォントの設定を.mlterm/fontに記述します。利用できるフォントはxlsfontsコマンドで調べることができます。例えば韓国語のフォントを調べる場合、韓国語の文字集合は「KS C 5601」なので「*ksc5601*」を付けて実行します。そうすると実行例Cのように利用できる韓国語のフォントの一覧が表示されます。同じフォントが何個も出てきて見にくい場合は、そのコマンドの後ろに「| uniq」を付ければよいでしょう。こうやって調べたフォントを設定ファイルに記述していきます。筆者が行っている設定例をリストAに示します。この例ではフォントサイズ16のものしか設定していませんが、さまざまなサイズを複数指定することもできます。

次に.mlterm/mainにメインの設定を行います。UCSフォントを使わない場合は次の設定を行います。

```
not_use_unicode_font=true
```

利用するフォントサイズを指定します。

```
fontsize=16
```

利用する文字符号化方式を指定します。利用するプログラムがすべてUTF-8対応であれば、ここに「UTF-8」と指定してもよいですが、現状ではそうでないため「EUC-JP」と指定しておきます。

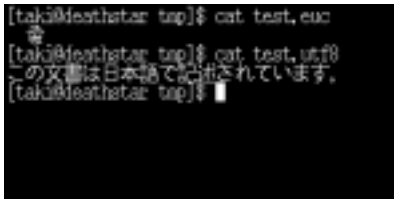
```
ENCODING=EUC-JP
```

Muttを起動する際にmltermの-Eオプションで UTF-8を指定すればよいでしょう。これで最低限必要な設定が出来ました。残りの設定は好みに応じて行ってください。

ここでちょっと実験をしてみましょう。まず、従来の環境で日本語の文書をEUC-JPで記述したファイルtest.eucを作成してください。実行例Dに、iconvコマンドを使ってUTF-8に変換する例を示します。従来の環境でこの2つのファイルを表示してみると画面Aのようになり、test.utf8の方が文字化けします。mltermを「-E UTF-8」オプションを付けて起動した場合では画面Bのようになり、test.utf8のファイルは正常に表示されるはずです。（滝澤隆史）

【画面A】従来の環境でtest.utf8を表示した例

【画面B】mltermを「-E UTF-8」オプション付きで起動した場合の表示例



【実行例C】test.eucをiconvコマンドでUTF-8に変換する

```
$ iconv -f euc-jp -t utf-8 test.euc > test.utf8
```

【実行例D】韓国語のフォントを調べる

```
$ xlsfonts *ksc5601*
-daewoo-gothic-medium-r-normal--0-0-100-100-c-0-ksc5601.1987-0
-daewoo-gothic-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
-daewoo-mincho-medium-r-normal--0-0-100-100-c-0-ksc5601.1987-0
-daewoo-mincho-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
-daewoo-mincho-medium-r-normal--24-170-100-100-c-240-ksc5601.1987-0
```

【リストA】筆者の.mlterm/fontの記述

```
ISO8859_1=16,-misc-fixed-medium-r-normal--16-*-iso8859-1
ISO8859_1_BOLD=-16,misc-fixed-bold-r-normal--16-*-iso8859-1
JISX0201_ROMAN=16,-misc-fixed-medium-r-normal--16-*-jisx0201.1976-0
JISX0201_ROMAN_BOLD=16,-misc-fixed-bold-r-normal--16-*-jisx0201.1976-0
JISX0201_KANA=16,-misc-fixed-medium-r-normal--16-*-jisx0201.1976-0
JISX0201_KANA_BOLD=16,-misc-fixed-bold-r-normal--16-*-jisx0201.1976-0
JISX0208_1978=16,-misc-fixed-medium-r-normal--16-*-jisx0208.1983-0
JISX0208_1978_BOLD=16,-misc-fixed-bold-r-normal--16-*-jisx0208.1983-0
JISX0208_1983=16,-misc-fixed-medium-r-normal--16-*-jisx0208.1983-0
JISX0208_1983_BOLD=16,-misc-fixed-bold-r-normal--16-*-jisx0208.1983-0
JISX0208_1990=16,-misc-fixed-medium-r-normal--16-*-jisx0208.1990-0
JISX0208_1990_BOLD=16,-misc-fixed-bold-r-normal--16-*-jisx0208.1990-0
JISX0213_2000_1=16,-misc-fixed-medium-r-normal--16-*-jisx0213.2000-1
JISX0213_2000_2=16,-misc-fixed-medium-r-normal--16-*-jisx0213.2000-2
KSX1001_1997=16,-daewoo-mincho-medium-r-normal--16-*-ksc5601.1987-0
GB2312_80=16,-isas-song ti-medium-r-normal--16-*-gb2312.1980-0
```

●ncurses

ncursesライブラリ（[5]）の場合は、執筆時点での最新バージョンである5.2に対して、さらに最新のパッチを当てたものであれば、UTF-8を扱えるようです。しかし、筆者が試したところ、Muttのconfigureスクリプトがワイド文字対応のncurseswの検出に失敗してしまったため、今回は確認できませんでした。

●ターミナルエミュレータ

UTF-8対応のターミナルエミュレータには、xterm（XFree86 4.0以降、[6]）やmlterm（[7]）がありますが、mltermの方が何かと便利なので、本記事ではmltermを使うことを前提として話を進めます。mltermについて知らない人は、コラム「mlterm」をご覧ください。

●エディタ

Muttでは、表示用の文字符号化方式が、そのまま文書の作成や編集用の文字符号化方式になります。そのため、UTF-8のファイルの読み書きに対応したエディタが必要です。

普段使用しているエディタがUTF-8に対応していれば、UTF-8で保存するような設定を行ってそのまま使えばよいでしょう。

UTF-8に対応していないエディタを使用している場合は、対応しているエディタに変えるという方法もありますが、好きなエディタを使えるというMuttの利点が失われます。この場合は文字コードを変換するラッパースクリプト経由で好きなエディタを起動させればよいでしょう。ラッパースクリプトに関してはコラム「ラッパースクリプト」をご覧ください。

●auto_viewで用いるプログラム

text/htmlのパートを表示するためにw3mを利用している方も多いと思います。しかし、これらをUTF-8で出力させるためには、国際化／マルチリンガル化w3m「w3m-m17n」（[8]）を使う必要があります。

また、多国語化ファイルビューア「lv」（[9]）が文字コード変換フィルタとしても機能しますので用意しておきましょう。参考までに.mailcapの例をリスト1に示します。w3m-m17nを使う場合は、urlviewの設定の変更も忘れずに行ってください。

●UCSフォント

日本語対応のUCSフォントには、Markus Kuhn氏が取りまとめたフォント（[10]）があります。xtermでUTF-8を使う場合はこのフォントが必要です。しかし、CJKを含めた多言語環境で使おうと思っても、フォントサイズは1つしかありません。そのため、現状では従来のフォントがそのまま使えた方が便利です。本記事で使用するターミナルエミュレータのmltermは、従来のフォントでもUTF-8が扱えるため、UCSフォントはなくても構いません。

●Input Method

Muttの最下行にあるラインエディタにはUTF-8で入力する必要があります。そのため、UTF-8に対応したInput Methodが必要です。しかし、mltermを使う場合は、EUC-JPで入力してもUTF-8に変換して入力してくれるため、UTF-8に対応したInput Methodは必要ありません。

また、メッセージの作成と編集に使うエディタでは、エディタがUTF-8での入力を要求していない限りInput MethodがUTF-8対応である必要はありません。

Muttの基本設定

●インストール

Muttの基本設定としては、先ほど述べたS-Langと静的にリンクすること以外は通常のMuttのインストールと同じです。ただし、mltermでUCSフォントを使わずに、従来のフォントのみを使う場合は、glibc-2.2やlibutf8を使用している場合でも、Mutt内蔵のワイド文字関数を使うようにconfigureのオプションとして「--without-wc-funcs」を必ず指定してください。理由は次で説明します。

●フォントの幅

私達が普段使っている漢字の文字集合「JIS X 0208」の固定ピッチフォントの幅は2桁です。そのフォントを使うEUC-JPのlocaleでも、ワイド文字関数は2桁として扱っています。ただしUCSフォントでは、JIS X 0208に含まれる文字の中で、記号類、ギリシャ文字、キリル文字など漢字以外の文字のフォントの幅は1桁になります。このUCSフォントを使うUTF-8 localeでも、ワイド文字関数は1桁として扱われています。

ここで問題となるのは、UTF-8 localeにおいて、JIS X 0208のフォントを使った場合、フォントの幅は2桁なのに、ワイド文字関数は1桁として扱うために、文字幅が一致しないことです。結果として、フォントの位置がずれるなど表示が乱れることになります。

そのため、この文字幅を、利用する環境に合わせて返すようにMutt内蔵のワイド文字関数を日本語パッチで拡張した設定変数$use_jisx0208を用意しています。この変数の値が「yes」であれば、JIS X 0208の記号類などを2桁の幅として扱い、「no」であれば1桁として扱います。ただし、これはmutt-1.3.27i-ja0.1.betaからの拡張機能です。また、Cライブラリやlibutf8のワイド文字関数を使った場合は補正できないので、この設定変数は無効になります。

そういうわけで、mltermでUCSフォントを使わない場合は、Mutt内蔵のワイド文字関数を使うようにして、Muttの設定に次の行を追加します。

```
set use_jisx0208=yes
```

●$charsetの設定

MuttでUTF-8 localeを使うためには設定ファイルに次の設定を行います。

```
set charset=utf-8
```

●$send_charsetの設定

送信に使用する文字符号化方式を$send_charsetに設定します。「us-ascii」を先頭に記述することを除けば、残りは自由に設定して構いません（リスト2）。

●$assumed_charsetの設定

受け取ったメッセージの文字符号化方式の自動判定を行うために、$assumed_charsetを設定します。ここには文字符号化方式の指定がないことが多いものを指定します。ただし、自動判定がうまくいくように、記述する順番には注意する必要があります。

記述する順番は、8ビットコードを含まないものを最初に指定し、その後は8ビットコードが多く含まれる順に指定します。具体的には、ISO-2022-*があったら、まずそれを先頭に記述します。次にUTF-8を記述します。その次にEUC-*があればそれを記述します。このとき

【リスト1】.mailcapの例

```
text/enriched; lv -Ou8 %s |richtext -t; copiousoutput
text/html; w3m-m17n -O utf-8 -dump -T %t %s; copiousoutput
```

【リスト2】$send_charsetの設定例

```
set send_charset="us-ascii:iso-8859-1:iso-2022-jp:euc-kr"
```

「EUC-JP」、「EUC-KR」、「GB2312」などは違いが判別できないため、どれか1つしか設定できません。最後に、ISO-8859シリーズの中から1つ記述します。設定例をリスト3に示します。

●$file_charsetの設定

添付するファイルの中でよく使われるものを指定します。記述する順番に関する注意事項は$assumed_charsetと同じです。ただし、UTF-8を必ず含めるようにしてください。設定例をリスト4に示します。使用する言語の種類が多いようでしたら、各種フックコマンドを使って、言語ごとに個別に指定した方がよいでしょう。

●$editorの設定

UTF-8対応のエディタを使用する場合は、$editorにエディタの名称を記述します。UTF-8対応でない場合は、前述したラッパースクリプトを記述します。

```
set editor="editor-utf8conv %s"
```

複数の言語を使い分ける場合には、使用する言語ごとに各種フックコマンドを使って、ラッパースクリプトを指定した方がよいでしょう。詳しくは「言語ごとの設定例」の章で説明します。

●その他

その他残りの設定については、今まで使っていた設定をそのまま使用してください。

また、Muttで使用する設定ファイルや署名ファイルなどに、日本語などによる記述がある場合は、そのファイルの文字コードをiconvコマンドなどでUTF-8に変換してください。

Muttの起動

●準備

まず、UTF-8を使うために次のようにlocaleを設定しましょう。

```
$ LANG=ja_JP.utf8
$ export LANG
```

次に、ターミナルエミュレータをUTF-8で表示するようにオプションを付けて起動します。mltermの場合は次のようにします。

```
$ mlterm -E UTF-8
```

このときにlocaleがないと怒られるようでしたらlocaleを作成してください。

mlterm上のUTF-8 localeを使用しているのであれば、この中からMuttを起動させるのもよいですが、そうでない場合はこの一連の作業を記述したシェルスクリプトを作って起動させた方がよいでしょう。その例をリスト5に示します。

Column ラッパースクリプト

UTF-8を扱えないエディタをMuttで使用する場合、UTF-8のファイルをEUC-JPなどに変換してからエディタに読み込ませ、編集が終わったらUTF-8に戻してあげる必要があります。

この一連の作業を行うラッパースクリプトの例をリストBに示します。この例では、「# -- editor --」のセクションに記述してあるように、jedをmlterm経由で起動させています。jedのようにターミナルエミュレータ上で動くエディタを直接起動すると、Muttが起動しているUTF-8 localeのターミナルから起動しますが、jedはUTF-8対応でないため使えません。そのため、エディタが利用できる文字符号化方式(例えばEUC-JPなど)に対応したターミナルエミュレータ(この例ではmlterm)経由で起動させます。一方、xjedのようにX上で動くエディタであれば、ターミナルエミュレータ経由ではなく、直接起動するようにすればよいでしょう。 (滝澤隆史)

【リストB】文字コード変換するラッパースクリプト経由でエディタを起動する

```
#!/bin/sh
ENCODING=EUC-JP
FILENAME=$1
TEMPFILENAME=${FILENAME}.$$

if [ -z "$FILENAME" ];then
  echo "Usage: editor-utf8conv FILENAME" 1>&2
  exit 1
fi
if [ ! -f "$FILENAME" ];then
  echo "editor-utf8conv: $FILENAME: No such file" 1>&2
  exit 1
fi

iconv -f utf-8 -t ${ENCODING} ${FILENAME} > ${TEMPFILENAME} || exit $!

# -- editor --
mlterm -E ${ENCODING} \
  -e jed ${TEMPFILENAME} -f set_buffer_no_backup
#xjed ${TEMPFILENAME} -f set_buffer_no_backup
# -- editor --
iconv -f ${ENCODING} -t utf-8 ${TEMPFILENAME} > ${FILENAME} || exit $!
rm ${TEMPFILENAME}
exit 0
```

【リスト3】$assumed_charsetの設定例

```
set assumed_charset="iso-2022-jp:utf-8:euc-kr:iso-8859-1"
```

【リスト4】$file_charsetの設定例

```
set file_charset="iso-2022-jp:utf-8:euc-jp:shift_jis"
```

【リスト5】localeを設定しつつmltermでmuttを起動するスクリプトの例

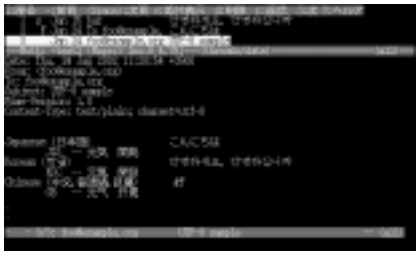
```
#!/bin/sh
LANG=ja_JP.utf8
COLORFGBG="white;black"
export LANG COLORFGBG
mlterm -g 80x40 -E utf-8 -e mutt "$@" &
```

●起動

以上でUTF-8を使うための環境が整いました。ここでMuttを起動してみてください。普段使っている通りに使えるはずです。参考までに、複数の言語が混在したメッセージを表示した例を画面1に示します。

なお、言語を使い分けてメッセージを作成する場合で、UTF-8対応でないエディタを使っている場合は、次の「言語ごとの設定例」の章に示すような設定を行う必要があります。

【画面1】複数の言語の混在したメッセージの表示



言語ごとの設定例

ここでは、メッセージを作成するときに使用するエディタの変更方法や、署名を変えるときの設定例を紹介します。

●フォルダごとの設定

procmailなどでメーリングリストごとに振り分けを行っている場合には、使用する言語はたいていフォルダごとに決まっています。そこでフォルダごとに、使う言語の設定を行うことを考えてみます。この場合には`folder-hook`を使って設定を変えます。

リスト6に`folder-hook`を用いた設定例を示します。この例ではデフォルトは日本語用の設定を行い、「`ko`」というフォルダでは韓国語用の設定を行い、「`en`」というフォルダでは英語用の設定を行います。ここで、`$assumed_charset`や`$file_charset`などを指定するのもよいでしょう。

●メッセージごとの設定

日本語のメッセージに対しては日本語で返信し、韓国語のメッセージに対しては韓国語で返信したいと普通は考えるでしょう。この場合は`message-hook`を使って、ヘッダの`Content-Type`フィールドに記述されている`charset`パラメータによって設定を変えます。

リスト7に`message-hook`を使った設定例を

Column　Muttも歩けば棒に当たる

Linux Japan愛読者の皆様、はじめまして。高橋全（たもつ）と申します。Kondara MNU/Linux版のRPM作成に少し参加しています。

私は1999年末ころからMuttを利用しております。そのころは1.0preでした。この号が出るころにはキャッシュ機能が実装されたMuttが出ているかもしれませんが、私のMuttに対する第一印象は「フォルダ移動が遅い！」でした。それは執筆時点でも少し気になっていることですけど、それでも使い続けているのですから、Muttの能力のほどがお分かりいただけるでしょう。

■変なヘッダが楽

本題に入ります。ここからは自分の`~/.muttrc`とそのソースから、皆様のお役に立てるかもしれない部分を探してご紹介します。

まず`my_hdr`です。X-*フィールドを簡単に追加できるのが便利です。私は**リストC**のようにしてPGP公開鍵の場所を書いたりしています。gnupg.orgのメーリングリストなどを見ると、X-PGP-*フィールドにもいろいろな書き方があるようですが、私はmutt-gnupg-howto（[11]）に書いてある通りにしました。

また、GIMPなどで白黒画像を作り、XEmacsのユーティリティでX-Faceにして保存し、**リストD**のようにして使用しています（できたデータ文字列のうち、一部の記号にはバックスラッシュを付けるなどの修整が必要なため、少し面倒ですが）。X-Faceは`unignore`（設定済みのメールヘッダを隠す設定）に含めておいて、pager画面に意味不明な記号が並んでいるときに見るようにするため、**リストE**のようにしています。これでjfbterm（またはkon2）上でも便利に使えています（[12]）。

また、私が利用しているNomailは、X-Moeパッチ*4を当てると簡単に日本語のX-*フィールドを付加できるようです。執筆時点でX-Moeパッチは配布元が「Not found」となってしまいますが、執筆時点のKondaraのCVSにあるnomail-0.4.10はX-Moe対応にしてあります（[13]）。Nomailにはメールキューを確認する`mailq`コマンドが付属しないので、自分で簡単な`mailq`を作って`~/bin`に置いています。`alias`でも可能でしょう。私のは「`root`権限で`grep`するだけ」というひどい代物なので見せられませんが、何かのキーに`macro`で割り当てると便利だと思います。

■メール看破

脱線しましたが、次は`to_chars`です。デフォルトだと日本人にはほとんど無意味かもしれませんが、「`>=-<-`」（最初の1文字はスペース）などのように、自分で見やすい文字を設定すれば、index画面で重要度が一目で分かって便利です。

こういう機能は重宝します。私はAPOPでサーバから`~/Maildir`に直接受信してしまうため、メーリングリストの議論が白熱しているときに何日も風邪を引いたりしていると、1カ所に数百通も集中してしまうからです。

実際は`limit`を使って表示を絞り込むことも多いのですが、パッと見ただけで分かるように、さらに`color`も変えておけば完璧です。振り分けは塩崎様（本誌2001年2月号掲載のコラムに登場）と同じように、ワンキーマクロで最後に確定、としています。

■hook漬け

Muttは柔軟な設定が売りなので、ついつい凝ったことをしてしまい、hookは増える一方です。「`In-Reply-To`」のないメールのために`folder-hook`で`strict_threads`を解除したり、`message-hook`で`content-type`に合わせてattributionを変えたり、`send-hook`で相手に応じてsignatureを変えたりしています。そのsignatureも、Muttを呼び出す`~/bin/mutt`を**リストF**のようにしておいて、私用メールの署名に自作fortuneをランダムに付加しています。

■config-sample

最後にKondara特有の話題を少々。

Kondara（2.1以降）には`config-sample`があるので、初めて利用するときにも`/usr/share/config-sample/mutt`以下にあるファイルを`~/.muttrc`などにコピーして少し編集するだけで「使える」状態になるはずです。

また、現在は変更されている可能性もありますが、コピーしなくても、`~/.muttrc`の冒頭に**リストG**を追加すれば日本語関連などの設定ができるようになっているはずです。しかし本来はコピーして編集するためのものですから、注意してご利用ください（さもないと自分の名前がVaderになってしまうかもしれません）。

■ コマンドを使う

以上のように`~/.muttrc`でいろいろと設定できるのも強みですが、とっさの作業にこそMuttの便利さが出ると思います。それで、正規表現を覚えたり`sed`などのコマンドを使いこなすことが鍵を握っています。逆に言えば、皆様のようにスキルをお持ちの方々にとってはMuttが「痒い所に手の届くMUA」になるということです。未経験の方は、この便利なMUAを一度試してみることを強くお勧めします。　（高橋全）

【リストC】my_hdrに記述したPGP公開鍵の場所の指定

```
my_hdr X-PGP-Key: http://www.jade.dti.ne.jp/~arms405/pubkey.asc
```

【リストD】my_hdrに記述したX-Faceの指定

```
my_hdr X-Face: `cat ~/.mutt/compface.txt`
```

【リストE】X-Faceの表示／非表示を切り替えるための設定

```
message-hook .            'macro pager \ef "!clear;echo no face\n"'
message-hook '~h x-face' 'macro pager \ef "|~/bin/view-x-face\n"'
```

【リストF】自作fortune付き署名の設定

```
#!/bin/sh
cp -f ~/.signature ~/.fortunesig
fortune ~/fortunes >> ~/.fortunesig
Eterm -e /usr/bin/mutt
```

【リストG】Kondara MNU/Linux 2.1以降でMuttの日本語関連の設定を行う設定

```
$ source /usr/share/config-sample/mutt/dot.muttrc
```

＊4 「X-Moeフィールド」を利用するためのパッチ。詳しくは`http://www.x-moe.org/`を参照。

示します。この例では文字符号化方式が「ISO-2022-JP」であるメッセージに対しては日本語用の設定を行い、「EUC-KR」であるメッセージに対しては韓国語用の設定を行い、「US-ASCII」や「ISO-8859-1」であるメッセージに対しては英語用の設定を行います。デフォルトでは日本語用の設定を行います。ただし、folder-hookが設定されている場合はその設定の内容をすべてmessage-hookのデフォルトの設定で上書きされてしまうので1行目のデフォルトの設定を取り除いてください。

●宛先ごとの設定

普段メールをやり取りする人の使っている言語はたいてい決まっています。そのため、宛先ごとに言語を設定することを考えてみます。この場合はsend-hookを使って、宛先によって設定を変えます。

リスト8にsend-hookを使った設定例を示します。この例では、デフォルトでは日本語用の署名ファイルとエディタを扱い、「foo@kr.example.org」という宛先に対しては韓国語用の署名ファイルとエディタを扱います。ただし、folder-hookやmessage-hookが設定されている場合、その設定の内容はすべてsend-hookのデフォルトの設定で上書きされてしまいますから、1行目のデフォルトの設定は取り除いてください。

●その他

さまざまな種類のフックコマンドを設定すると、その設定が上書きされてしまい、思ったとおりの動作にならないことがあります。もちろん、ここで紹介した設定例をすべて記述した場合もそうです。特にデフォルトの設定には注意してください。

なお、フックコマンドに関して詳しく知りたい方は本誌2001年9月号の記事をお読みください。

最後に

記事を読んでもらうと分かるとおり、多言語のメール環境を構築するには、Muttそのものよりも、その周辺の環境を整えることが主な作業となります。現状では、さまざまなものがUTF-8対応への過渡期である状態ですが、2年くらいすると環境もかなり整ってくると思います。しかし、欧米の人たちはどうも「UTF-8にさえ対応すれば国際化、多言語化はOKさ」と思っているような節があるので、日本のコミュニティもそれぞれの開発プロジェクトに積極的に参加していく必要があります*5。

【リスト6】フォルダ別に使用言語を分ける設定

```
folder-hook . 'set signature=~/.mutt/signature-ja editor="editor-ja %s"'
folder-hook ko 'set signature=~/.mutt/signature-ko editor="editor-ko %s"'
folder-hook en 'set signature=~/.mutt/signature-en editor="editor-en %s"'
```

【リスト7】メッセージ別にフォルダを振り分ける設定

```
message-hook . 'set signature=~/.mutt/signature-ja editor="editor-ja %s"'
message-hook '~h "^content-type:.*iso-2022-jp"' \
  'set signature=~/.mutt/signature-ja editor="editor-ja %s"'
message-hook '~h "^content-type:.*euc-kr"' \
  'set signature=~/.mutt/signature-ko editor="editor-ko %s"'
message-hook '~h "^content-type:.*(us-ascii|iso-8859-1)"' \
  'set signature="~/.mutt/signature-en" editor="editor-en %s"'
```

【リスト8】宛先別に署名ファイルとエディタを振り分ける

```
send-hook . 'set signature=~/.mutt/signature-ja editor="editor-ja %s"'
send-hook foo@kr.example.org$ \
  'set signature="~/.mutt/signature-ko" editor="editor-ko %s"'
```

Resource

[1] Mutt Japanese Edition
http://www.emaillab.org/mutt/

[2] libutf8
ftp://ftp.ilog.fr/pub/Users/haible/utf8/

[3] S-Lang
http://www.s-lang.org/

[4] Unicode Mutt
http://www.rano.org/mutt.html

[5] NCURSES
http://dickey.his.com/ncurses/ncurses.html

[6] xterm
http://dickey.his.com/xterm/

[7] MLTERM
http://mlterm.sourceforge.net/index.ja.html

[8] w3m-m17n
http://www2u.biglobe.ne.jp/%7Ehsaka/w3m/index-ja.html#m17n

[9] LV Homepage
http://www.mt.cs.keio.ac.jp/person/narita/lv/index_ja.html

[10] Unicode versions of the X11 "misc-fixed-*" fonts
http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts.tar.gz
http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts-asian.tar.gz

[11] Everything You Need To Know To Start Using GnuPG with Mutt（日本語訳）
http://www.linux.or.jp/JF/JFdocs/mutt-gnupg-howto/

[12] The Mutt mailreader「view-x-face」
http://www.spinnaker.de/mutt/view-x-face

[13] Kondara MNV/Linux 版 Nomail
http://furi.kondara.org/cvsweb/cvsweb.cgi/Kondara/pkgs/STABLE/nomail

＊5　日本特有の事情が入りすぎて嫌がられ、受け入れられないこともあります。それがMutt 1.3系列の日本語パッチができたきっかけだったりします。当初、本家にすべて入れようと思っていたのですけどね。