1995 年 夏のプログラミングシンポジウム　　1995-07-26
Mobile & Ubiquitous Computing　　@NTT ゆかり荘

携帯電話と携帯情報端末でインターネットメイルを読む

富士通研究所
和田英一
(wada@u-tokyo.ac.jp)

ことのおこり

最近割合とルンルン気分なのはインターネットに来たメイルが自宅で読めるようになったことである. もちろん以前から自宅でもメイルの読み書きのできる人はたくさんいた. つまり, ポータブルのパソコンを自宅におき, モデムで大学に接続してやっていた.

私の場合は, 自宅の電話が親子になっており, モデムが簡単にはつかない; ポータブルといってもパソコンは結構重い; 日曜にまでメイルを読まなくても別に不便はない; などの理由でこれまで自宅でメイルを読む努力はしなかった.

しかし, 企業の研究所に勤めるようになると, 土曜, 日曜が休みだし, 木曜, 金曜と出張があったりすると, 月曜にもう少し早く読むべきであったメイルを見つけたりして, 何とかしようかと思うようになってきた. ところが我が家の電話環境は前と同じだし, よしモデムがあっても, 企業の研究所は大学と違ってガードが堅く, 外から簡単にはログインできないという条件もある.

私の場合は, 他にもアカウントがあるので, そちらでメイルを受信すれば, そこへログインしてメイルが読めるのだが, 普段あまり読まないアカウントにメイルをフォワードしていると, そこのスプールがパンクする危険がある. かといって, cron を動かしてときどき捨てるというのも気が進まず, これもやる気にならなかった.

インターネットのメイルを Nifty へ転送する

では, 解決策は何かというと, インターネットにきたメイルをリモートコントロールして, Nifty で読むのである. その前に私宛のメイルは通常どういう経路でながれてくるかを説明しなければならない. 実は下のようになっている. (Figure 1, 東大, Nifty, 富士通, wide などのメイルアドレスを使っている.)

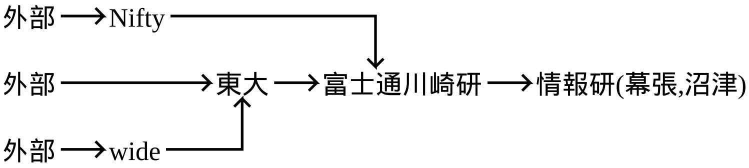

Figure 1

メイルのスプールは川崎研と情報研 (幕張と沼津) の計 3 箇所にある．これを自宅で (あるいはどこか別の場所で) 読むにはまず Nifty からインターネット (川崎研) への回送を解除する．それには Nifty に入り，電子メイルのメニューにいき，その他→メイルオプション→自動回送解除とまわれば解除できる．そしてインターネットのスプールにあるメイルを Nifty へ転送するのである．(Figure 2)

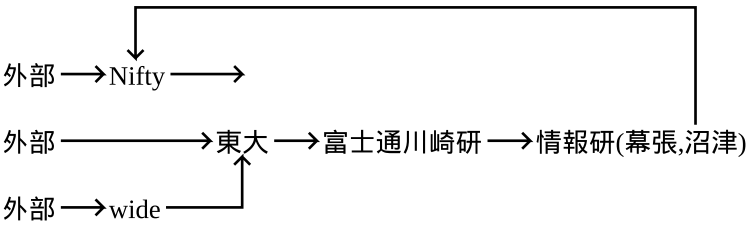

Figure 2

lispagent の設計

いろいろ考えた上で私は，自分が差出人で特定の subject を持ったメイルを受信すると，その本文を Lisp のプログラムと見て次々と評価 (`eval`) し，その結果を送り返すプログラムを用意した．それには unix のソフトで `procmail` というものがあり，それが沼津の研究所の環境にあったので，それを利用する． ホームディレクトリに `.procmailrc` なるファイルを用意しておくと，メイルを受信するたびに `.procmailrc` をしらべ，そこに書いてある条件に合うと，その続きに書いた作業をしてくれる．これは私の知らないうちに，私の身代りになって走ってくれるシステムなので agent，それも lisp を評価するので lispagent と呼ぶ．

`.forward` と，これで起動される `.procmailrc` は次のようにできている．

```
w@amulet(92) cat .forward
"|IFS=' ';exec /usr/local/bin/procmail #ew"
w@amulet(93) cat .procmailrc
```

```
PATH=/bin;/usr/bin;$HOME/bin/utilisp
MAILDIR=$HOME/Mail
#DEFAULT=$MAILDIR/mbox
#LOGFILE=$MAILDIR/from
:0 c
* ^Subject.*xxxxxxxxxxxxxxxxxxxx
* ^From.*wada@u-tokyo.ac.jp
| cat > $HOME/mailagent/standin; cd $HOME/mailagent;
$HOME/bin/utilisp -F $HOME/mailagent/niftyfilter.l;
$HOME/bin/utilisp -F $HOME/mailagent/niftyhandler.l
>$HOME/mailagent/standout; $HOME/bin/utilisp
-F $HOME/mailagent/sendback.l
```

.procmailrc の下の方を説明しよう. `* ^Subject.*xxxxxxxxxxxxxxxxxxxx` と `* ^From.*wada@u-tokyo.ac.jp` はメイルに Subject が xxxxxxxxxxxxxxxxxxxx(極秘), From が wada@u-tokyo.ac.jp とあれば, 以下のシェルスクリプトを起動するという意味である. シェルスクリプトの中では, まずメイルの内容を standin というファイルにコピーし, つぎに mailagent へ cd し, utilisp を起動して niftyfilter.l というファイルを食べさせる. niftyfilter.l の役目はメイル (と PEM) のヘッダーを取り除くことである. 結果は standin1 というファイルにできる.

```
w@amulet(95) cat niftyfilter.l
(defun niftyfilter nil
   (lets ((err:end-of-file #'(lambda (x (y)) (throw 'loop)))
          (break #'(lambda nil (quit)))
          (path (string-append (getenv "HOME") "/mailagent/"))
          (s (stream (string-append path "standin")))
          (s1 (stream (string-append path "standin1"))) (line))
    (inopen s)
    (outopen s1)
    (catch 'loop
       (loop (and (/0= (string-length (readline s))) (exit)))
       (loop (setq line (readline s))
            (cond ((and (= (string-length line) 40)
                        (string-search
                         "-----BEGIN PRIVACY-ENHANCED MESSAGE-----"
```

```
                                           line))
                                     (loop (and (/0= (string-length (readline s)))
                                                      (exit))))
                                   ((and (= (string-length line) 38)
                                         (string-search
                                          "-----END PRIVACY-ENHANCED MESSAGE-----" line)))
                                   (t (princ line s1) (terpri s1)))))
    (close s1)
    (quit)))
(niftyfilter)
```

つぎの niftyhandler.l は niftyfilter.l が standin1 に用意したファイルを lisp のプログラムと思って実行するものである. 実行の結果は standout のファイルに書く.

```
w@amulet(96) cat niftyhandler.l
(exfile (string-append (getenv "HOME") "/mailagent/standin1") t)
(quit)
```

シェルスクリプトの最後は standout にできている実行結果のファイルを toaddress にあるアドレスに送り返すプログラム, sendback.l を utilisp で動かす.

```
w@amulet(97) cat sendback.l
(defun niftysendback nil
   (lets ((break #'(lambda nil (quit)))
          (path (string-append (getenv "HOME") "/mailagent/"))
          (s (inopen (stream (string-append path "toaddress"))))
          (toaddress (prog1 (read s) (close s))))
    (call (string-append "cat " path "standout|/usr/ucb/mail "
           "-s ""mailagent reply"" " toaddress))
    (quit)))
(niftysendback)
```

Lisp はもちろん自宅の Macintosh でも, Oasys pocket 3 でも実行できるが, 上述の機能を使えば沼津で Lisp を実行し, 結果が得られるようになった. 例えば

```
(defun fact (n) (cond ((zerop n) 1) (t (times n (fact (sub1 n))))))
(fact 100)
```

という本文のメイルを送ると, しばらくして私の Nifty のメイルボックスへ

```
fact
933262154439441526816992388562667004907159682643816214685929638952175999932
29915608941463976156518286253697920827223758251185210916864000000000000000000
00000000
```

といメイルが返ってくるのである.

もちろん相当自信のある完全プログラムを送らないと, しばらくしてエラーメッセージがもどり, がっかりする.

mailagent の構造

ところで私は`.procmailrc` や `lispagent` の用意ができると, 一連の Lisp 関数を設計して一つのファイルにまとめ, `procmail` の作業をするときのディレクトリにおいた. その主要な関数は `resetcounters` と `mailagent` である.

- `resetcounters` は作業環境を初期化する. 週末の自宅で, 最初に転送機能を使う時にはまずこの関数を送る.

- `mailagent` はメイルのスプールからまえの続きを読みだし, メイルに分解し, Nifty へ転送すべきかどうかを判断し, 転送すべきならバッファのディレクトリに追加する.

このようにスプールにある未読のメイルをすべて転送するのではなく, たとえば自分へ出したカーボンコピーのメイルは転送しない; また, さほど重要でないメイリングリストから来たメイルは転送しない; さらに 300 行をこえる長大なメイルも転送しないようにしてある. これらの条件はプログラムのなかに記述してあり, リモートで変更するのは容易でない.

バッファのディレクトリへ切りわけが済んだら, 規定の数になるまでのメイルをバッファからとりだし, 私宛に (Nifty またはインターネットへ) 送る. 最後に転送すべきメイルがバッファにあとなん通残っているかも送る. (送るべきメイルがないこともあり, その時はその旨のメイルを送る.)

この記述から判明するように, 週末の自宅から, 1 回めは `(resetcounters)` `(mailagent)` のメイルを送り, 2 回目からは `(mailagent)` だけのメイルを送ればいいが, これで送られてくるのは,

沼津のスプールに未読で残っているメイルの先頭から転送条件をクリアしたものである．これでは，ほかの研究所で既読 (だが沼津では未読) のメイルも転送されてくるので，週末の帰宅の前に沼津のスプールを捨てるという作業が実用上必要であった．

その代案として，関数 `(resetcounters)` にパラメータを付けられるようにした．`(resetcounters starttime)` の形となっており，`starttime` は `"xxx Jul 21 17:00:00 1995"`のような文字列である．`xxx` のところには曜日を書くのだが，曜日は比較に使わないので，任意の文字列を書いておく．

転送システムにはこの `starttime` 以降のメイルだけを処理させることとした．この機能で未読のメイルだけ抽出できるようになった．

こういうハックをしたと話すと，同僚には不思議そうな顔をするのがいる． インターネットに Nifty へのフォワード指令を書いておけば良いではないかというのである．

しかし，ことはさほど簡単でない．このようにするとすべてのメイルが転送されてしまう．さらに悪いのは，Nifty はセンターにメイルスプール (メイルボックス?) をもっているから，一人につきあまり多くのメイルを溜めておくことができない．(25 通くらいがリミットで，それを越えると差出人に送り返される.) そういう次第だから，インターネットに到着するメイルを闇雲に Nifty にフォワードするわけにはいかないのである．

使ってみたら

自宅の方はどうなっているかというと，仕掛けは Oasys pocket 3 と digital mova F で，Oasys pocket 3 は unix こそ動かないが MS-DOS が使えるから，Lisp とか nemacs とかは動き，なかなか調子がいい．しかし今の話題では MS-DOS は関係ない．

とりあえずは Oasys の autocom を利用して Nifty のメイルの読み書きをしている． autocom はメイルの送信予約をしておき，Nifty に接続しようとすると，自動ログインしたあと，予約してあるメイルを送信し，つづいて Nifty にきている未読のメイルをすべて読みとり，回線を切ってくれる．手動で Nifty を使うと，いちいち確認を聞いてきたりして，まどろっこしいが，autocom では眺めているうちに送受信がすんでしまう．

手元には 1 回め用のメイルの本文，2 回め以降用の本文，`starttime` つきでリセットするテンプレートのファイルが MS-DOS のファイルとしておいてあるので，まずこれらのいずれかを選んで送信予約の電文をつくる． 始めのうちはいちいち `Subject` を入力していたが，ファイルの 1 行目がデフォールトの `Subject` になることがわかったので，いまではファイルの 1 行目に `Subject`

を含む文字列を書いている．この文字列も Lisp で評価されるが，別に悪いことは引き起こさないので，快適である．

現在面倒なのは Nifty からインターネットへの回送を解除することで，これは autocom ではできないため (と断言はできないが，今はやり方がわからない)，研究所から帰る際に解除を忘れた時には，そのために一度自宅から Nifty へ手動で入る必要があるが，autocom のマクロを修正すれば，解除もついでにできるようになるのではないかと希望的に考えている．

だが，本当は自分で自家用 autocom を書きたい． モデムと通信する時のデバイス名がわかれば Lisp でプログラムくらい書けると思うのだが，いろいろ調べているものの，まだよくわかっていない．

思わぬトラブル

ところであるとき，`mailagent` のプログラムが暴走するという事件が起きた．沼津では `/var/spool/mail` のおいてある共用計算機の上で，これまで説明したプログラムが走っていたのだが，この計算機は管理者の主義として，使用できるファイル容量の上限が設定してなかった．procmail はメイルで起動されるから，プログラムを作成した筆者のモニターしていない時にも当然走り出す． そして，一定の仕事が済むと `(quit)` を実行して `utilisp` を終了，したがってシェルスルリプトからも抜けるのだが，あるときファイル容量のバランスをとるということで，ホームディレクトリの変更が行なわれた．ホームディレクトリは今でこそ，シェルスクリプトでは，上の例のように `$HOME`，`utilisp` では `(getenv "HOME")` と書いてあるが，最初は絶対パスで記述してあった．(まさかホームディレクトリが変わるとは予想していなかった.) その変更の結果，シェルスクリプトの中で指定したディレクトリが存在しないわけだからこのシステムか動かなくなったのは当然で，それからあわててあちこちのディレクトリの指定を変更して回った． その結果やっと機能は復活したように思えたが，未変更の箇所が残っていたのが致命的であった．

つまりあるプログラムが走った結果，ファイルが見つからないという事態が発生した． そこで，制御は `break` へ移り，標準の `break` ルーチンが `funcall` された． `utilisp` はインターラクティブに使うことを前提としている (UtiLisp = University of Tokyo Interactive Lisp という説もある) ので，当然 `terminal input` から読み込みを開始しようとするが，`end-of-file` になり，したがってふたたび `break` へいく．これを永久に繰り返した挙げ句，ついにファイルを使い切って計算機が落ちたのだ．

その後はさまざまな防護策を講じている．たとえば上で見たようなホームディレクトリに絶対パスを書かないとか，`break` を `quit` を呼ぶようにするとか (`(setq break #'(lambda nil (quit)))`)，示したプログラムでは省略したが，一定の秒数で `alarm` がかかって `quit` をかけるとかである．

セキュリティ問題その他

このシステムのことをこの 3 月の WIDE 合宿で紹介したところ, セキュリティの面で弱いのではないかという意見があった. Oasys から送るメイルに PEM をかけるというのはどうもオーバーな感じなので, せいぜい S/key の方式程度にしたいと検討中である. S/key はログイン用だから, 先方からチャレンジを聞いてくるが, 一方通行のメイルではそうもいかないので, あらかじめ使い捨てパスワードをいくつか手元に用意し, それを順番に使うということになるであろう. できれば `Subject` を特定のものにし, その後半に使い捨てパスワードを書いておくというのがいいが, そのためには `procmail` に機能を追加しなければならず, `procmail` 相当のプログラムを自分で書くか, `procmail` では `Subject` をチェックするだけで, 使い捨てパスワードは本文の先頭に書くことにし, `niftyfilter` で, メイルヘッダーを取り除いた時にパスワードチェックをするかになると思っている.

日本シティメディア (JCM) の無線モデム (TRM400) でもこのシステムを使ってみた. このサービスはまだ国道 16 号線の内側くらいでしか受けられないそうだが, 我が家は大宮で, 16 号線からほんのすこしはみ出しているだけだから, いささか心配だったが使えるようである. もっとも使用中にいきなり `NO CARRIER` とかでて, 急に切れてしまうことを幾度か経験したので, はみ出しのせいかとなかばあきらめのムードでもある.

このモデムでは autocom は使えないので, 通常のパソコン通信の Nifty アクセスになってしまう. したがって, こちらの出番は自動回送解除と第 1 回目のメイルの発送のみとなっている.