インターフェイスの街角 (65) — IMAP を利用した楽々メール管理

増井 俊之

これまで、私はずっと社内の POP サーバーと MH を使ってメールの読み書きをしていましたが、最近、レンタルサーバーを借りたのを契機に、IMAP を使い始めました。

そこで、今回は IMAP を便利に使うためのツールを紹介します。

IMAP

IMAP (Internet Message Access Protocol) は、サーバーにメッセージを保存したままメールを読むことのできるプロトコルです。

POP では、サーバーからメールを取得すると、基本的にメッセージがローカルマシンに転送されてしまうため、それ以降は別のマシンからそのメッセージは読めなくなります。一方、IMAP では、メッセージ本体がメールサーバーに残っているため、どのマシンからでもメッセージが読めるのが大きな特徴です。メッセージの読出しやメールボックス管理などの処理は、すべて IMAP 経由でリモートから操作できます。したがって、Web ブラウザでメールを読める Web メールシステムがあれば、いつでもどこでもメールを見られるので、たいへん便利です。さらに、Becky!などの Windows 上のメーラ (MUA) でも、Wanderlust のような Emacs 上のシステムでも、同じようにメールの読み書きができます。

プロトコルとしてみた IMAP は POP にくらべてはるかに複雑で、基本機能を定義した RFC2060 に加え、メールボックスの管理やメッセージの管理、認証、検索などについて、複数の RFC で数多くの機能が定義されています。

IMAP は、メールに関するあらゆる操作をリモートからおこなえるため、モバイル端末機器向けのプロトコルであると説明されていることもあります。しかし、〝モバイル〟が複数の機器からメールを読む機会が多いという意味であるとすれば、最近は自宅と会社の両方でメールを読むといったように、モバイルな状況がごく普通ですから、その点でも有用だと思います。

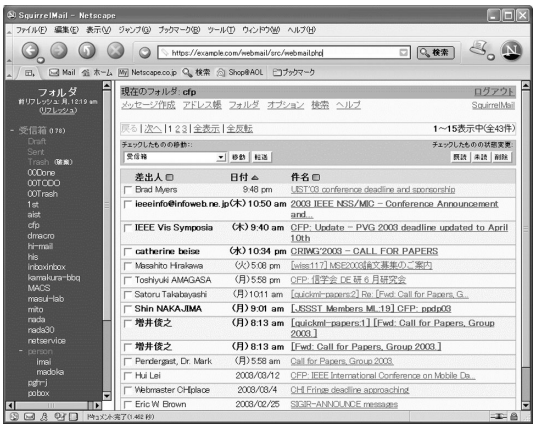
IMAP のプロトコルの詳細は、@IT のページ[1]や書籍[1]などで詳しく解説されています。

Web メール

Web ブラウザを利用してメールを読み書きできるシステムは、一般に Web メールシステムと呼ばれています。代表的なものに Netscape WebMail[2]や HotMail[3]などがあり、数多くのユーザーが利用しています。これらのシステムは独自のプロトコルを使っていますが、IMAP を利用する Web メールも IMP[4]や SilkyMail[5]、EMU-MAIL[6]などがあり、最近は SquirrelMail[7]の人気が高いようです。SquirrelMail は PHP で書かれた Web メールシステムで、IMAP の機能をフルに活用する多くの機能を備えています。メールの送受信やメールボックスの管理といった基本的な機能はもちろん、各種のプラグインで機能を拡張できるのが特徴です (図 1)。

1 http://www.atmarkit.co.jp/fnetwork/rensai/netpro08/netpro01.html
2 http://webmail.netscape.com/
3 http://www.hotmail.com
4 http://www.imp-jp.org/
5 http://www.cyrusoft.com/silkymail/
6 http://www.cyrusoft.com/silkymail/
7 http://www.squirrelmail.jp/

図 1　SquirrelMail のメールのリスト表示



Web メールシステムのメリットとして、どのような環境でも Web ブラウザさえあればメールの読み書きができるという点がよく挙げられます。しかし、Web ブラウザ上でメールを読める利点はそれだけではありません。

たとえば、さまざまな予定や地図などがメールで送られてくることはよくあります。メッセージを Web ページで見られるのなら、Web ページ上の予定表などに直接そのメッセージへのリンクを記入することができます。Web メールと Wiki があれば、個人情報管理システムやグループウェアなどで提供されている機能の大半は、普通の Web ブラウザ上で実現できてしまいます。

メールの本体が手許にないとちょっと不安なこともあるかもしれません。たしかに、簡単な Web メールでは、Web ページにアクセスできなければまったくメールが読めません。しかし、ローカルにキャッシュを保存するメーラを使えば心配ありません。このようなメーラでは、IMAP サーバー上のメールとローカルのキャッシュの同期がおこなわれるため、ネットワークへの接続手段がない環境でも手許でメールを参照できます。

IMAP と Web メールの併用

SquirrelMail を導入して快適な IMAP 生活を送れるようになり、しばらくはそのまま使っていたのですが、慣れてくると欲が出てきました。たとえば、以下のような点について不満を感じるようになってきました。

- メールの自動振分け機能

  メールの自動振分けについては、2002 年 10 月号の「横着プログラミング」に解説があります。しかし、IMAP に対応した方法で SPAM メールを破棄したり、メッセージを自動的に適切なメールボックスに振り分けけるシステムがあれば便利でしょう。メールの到着時に振分けをおこなうツールとしては、procmail や maildrop がよく使われているようです。ところが、これらのシステムでは振分け規則を特殊な記号や言語で記述しなければなりませんし、独自の振分けアルゴリズムを利用するのは難しいのが実情です。将来的なことも考えると、SPAM 以外のメッセージを振り分けて、バックアップ用のメールボックスに格納するプログラムを作ったほうがよさそうです。

- 検索への対応

  私が使っている Courier-IMAP[8] という IMAP サーバーでは、〝Maildir〟という形式でメールを管理しています。メールをファイルに格納する方式は、これまでにもいろいろと考案されてきました。代表的なものに、複数のメールを 1 つのファイルで管理する Mailbox 形式や、メールボックスをディレクトリで表現し、各メッセージを異なる番号をもつファイルとして管理する MH 形式があります。

  Maildir は、qmail で初めて採用された比較的新しい形式です。メールボックスごとに異なるディレクトリが使われ、メッセージごとに異なるファイルが使われる点は MH 形式と同じですが、時刻やマシン名などをファイル名の一部に使用するため、異なるメッセージはかならず異なるファイル名になることが保証されています。さらに、配送や転送の際の障害によってメールを失わないような工夫も加えられています。

  同じ Maildir 形式であっても、ディレクトリやファイルの命名法はサーバーごとに異なります。Courier-IMAP の場合は、〝example.com〟というマシン上にあるフォルダ〝abc〟に置かれたメールは、

```
~/Maildir/.abc/cur/1048261322.24758_0.⇒
example.com:2,RS
```

  のような名前のファイルとして格納されます（誌面の都合上、⇒ で折り返しています。以下同様）。ファイル名の先頭の数字は時刻を表し、末尾の〝2,〟はその

8 http://www.inter7.com/courierimap.html

後にフラグ文字が続くことを意味します。フラグは R (Replied) や S (Seen) などの文字で表現され、そのメッセージに対して返信したか、すでに読んだか、といった状態を表します。
このようなメールが ``def'' というメールボックスに移動されると、

```
~/Maildir/.def/cur/1048261322.24758_0.⇒
example.com:2,RS
```

のようにパスが変化しますし、さらに未読状態に変更したりすると、

```
~/Maildir/.def/cur/1048261322.24758_0.⇒
example.com:2,R
```

のようにファイル名まで変わってしまいます。
ファイル名の一部にメールの状態を示す文字があり、これがしばしば変化する可能性があるため、ファイル名に依存するシステムと併用すると問題が起こる可能性があります。たとえば 3 月号で紹介した全文検索システムは、ファイルのパスにもとづいて単語出現頻度ファイルを生成するため、パスが変わるたびにキャッシュを再計算しなければなりません。
ファイル名が変更されない検索専用のメールボックスにメッセージを格納しておき、検索時にはそれらのファイルを使うことにすれば、状態の変化によって検索システムの負担が増すといった事態は避けられます。前述の振分けプログラムなどを使い、あらゆるメッセージを保存しておけばいいでしょう。メールのバックアップがかならず保存されるのなら、誤ってフォルダ内のメールを削除した場合でも安全です。

- システムの重さ

  SquirrelMail は専用のメールクライアントに匹敵する機能を備えていますが、ソースコードは数万行もあり、実行はあまり高速とはいえません。また、フレームやフォームがたくさん表示されるため大きな画面が必要で、PDA や携帯電話からのアクセスには不向きです。
  外出中に緊急にメールをチェックしたい場合などは、機能を制限した簡易版 Web メールのほうが手軽に使えます。新着メールのチェックとメール送信に機能を絞った Web メールと併用したくなります。
- メッセージへのリンク

  SquirrelMail ではメッセージウィンドウの URL は

```
https://example.com/webmail/src/read_body.⇒
php?mailbox=INBOX&passed_id=123&start⇒
Message=1
```

のような形式で表現されます。数字の ``123'' などはフォルダの中身によって変化する可能性があるため、この形式の URL ではつねに同じメッセージを指すことはできません。
IMAP には検索機能があるので、ヘッダの Message-ID 情報などにもとづいてメッセージを特定できます。簡易版 Web メールでは、メッセージへの直接リンクを表現する URL も使えるようにしておくと便利でしょう。

そこで、これらに対応したメール振分けプログラムと、軽い Web メールシステムを作ってみることにしました。

lens ── メール振分けプログラム

メールはいろいろな用途にひろく使われているため、なんらかのかたちで複数のメールボックスに分類して管理している人が多いのではないでしょうか。一目で SPAM と分かるメールは届いた瞬間に消したほうが得策ですし、友人や家族などからのメールは差出人ごとに自動分類されると便利です。一方、時間の経過とともに重要度が変化するようなメールは、その時々の重要度に応じて手作業で分類したり、あとで再分類する必要が出てくるかもしれません。

From:や Subject:などのパターンを利用し、メールが届いたときでも、届いたあとでも振り分けられる **lens** というプログラムを作ってみました[9]。

メールが届いたときに自動的に振分け処理をおこないたいときは、自分のホーム・ディレクトリにある.forward ファイルを利用するのが一般的です。Sendmail や Postfix などのメール転送システム (MTA) は、メールを配送する前に各ユーザーの.forward ファイルを参照します。このファイルに、

```
% cat ~/.forward
masui@example.com
%
```

9 メールを自動分類するソフトウェアの分野では、「Information Lens」[2] というグループウェアの研究が有名です。lens という名前は、これにちなんだものです。

のようにメールアドレスが記述してあると、届いたメッセージをローカル・ファイルシステムに格納する代わりにそのアドレスに転送します。また、

```
% cat ~/.forward
| /usr/local/bin/lens
%
```

のようにプログラム名が記述されている場合は、メッセージを標準入力としてそのプログラムに渡すことができます。

引数を指定しない場合は標準入力に入力されたメッセージの自動振分けをおこない、引数を指定したときはそれに応じてメッセージを振り分けるような構造にしておけば、自動でも手動でもメッセージの振分けを指示することができます[10]。

lens は約 200 行の Ruby プログラムです。誌面に掲載するにはちょっと長いので、新たに開設した私の Web ページ[11]で公開します。

lens は、設定ファイル (~/.lens) に記述されたホスト名や転送アドレス、振分けパターンなどを読み取ったあと、振分け規則に従ってメッセージの配送や振分けを実行します。引数なしで起動された場合は標準入力をメッセージとして処理し、引数付きで起動された場合は引数で表現された日時より新しいメッセージに対して振分けをおこないます。たとえば、

```
% lens 3/10
```

として起動すると、3 月 10 日以降のメールについて振分け処理を実行します。

軽い Web メール

さきほども述べたように、SquirrelMail はプラグインも合わせると PHP のコードが数万行もあり、実行も軽快とはいえません。せっかくの Web メールなのに、携帯電話のブラウザでは使えませんし、PDA のブラウザでの利用もほとんど不可能です。新着メールを読んで最低限の返事をするといった基本的な機能に絞った Web メールがあれば、より快適に IMAP が使えます。

10 メール振分けの方式は、2002 年 10 月号の「横着プログラミング」に書かれているものとほぼ同じです。

11 http://www.pitecan.com/

リスト 1　lens の設定ファイル .lensrc

```
# Configuration for 'lens'

LensConfig = {
  :hostname => 'example.com',
  :smtp_host => 'localhost',
  :local_address => 'masui@example.com',
  :mobile_address => 'foobar@ezweb.ne.jp',
  :maildir => "#{ENV['HOME']}/Maildir",
}

class Message
  @@non_important_mls = {
    'enkai-talk'        => 'enkai-talk',
    'old-systems'       => 'system/old',
     # あまり重要でないメールの振分け
  }
  @@subject_patterns = {
    'ipsj'              => 'ipsj',
    'pobox'             => 'system/pobox',
     # Subject:行のパターンに応じた振分け
  }
  @@from_patterns = {
    'masui@'            => 'person/masui',
    '@csl.sony.co.jp'   => 'csl',
     # From:行のパターンに応じた振分け
  }

  @@spam_patterns = {
    'Content-type'      => [
      /gb2312/i
    ],
    'From'              => [
      /spammer/,
      ],
    'Subject'           => [
      / (viagra|adult) /i,
      # SPAMメールは破棄する
      ],
  }
end
```

同じことを考える人はいるもので、Ruby の IMAP ライブラリを用いた mongle[12] という小さな Web メールが公開されていました。Ruby の net/imap という IMAP ライブラリを利用すれば、IMAP の複雑なプロトコルを Ruby から操作できるので、簡単に Web メールシステムを作成できます。

mongle はごく軽い Web メールですが、それでも大きな画面を前提としているようです。末尾のリスト 2 の webmail.cgi は、携帯電話などの小さな画面にも対応でき

12 http://home.sunheartcowfork.com/mongle/

図 2　メールのリスト表示

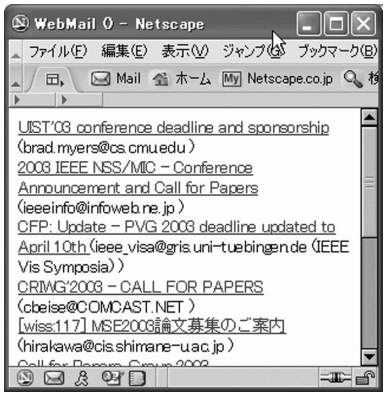
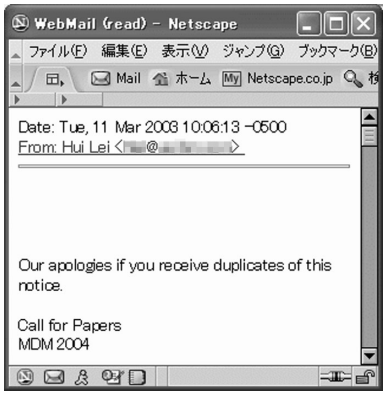


図 3　メッセージ表示



るように mongle を改造した Web メールシステムです。

図 2 は、フォルダ内のメールのリストを表示したところです。表示内容を絞っているので、これなら携帯電話でも十分実用になります。

図 2 でタイトルを選択すると、図 3 のようにメッセージが表示されます。差出人をクリックするとメール編集画面になり、メールを送信することができます。

webmail.cgi はわずか 120 行のプログラムですが、とりあえず Web メールシステムとして使えるのは IMAP と Ruby のライブラリの威力といえるでしょう。

webmail.cgi でメールの本文を読むときは、番号などでメッセージを参照するのではなく Message-ID から参照するようにしています。たとえば、Message-ID が〝xxyyzz〟のメールは以下のような URL で参照できます。

```
http://example.com/webmail.cgi?mailbox=INBOX&⇒
message_id=xxyyzz&cmd=read
```

webmail.cgi の引数に〝read〟を指定すると、IMAP の SEARCH コマンドで Message-ID を検索し、メッセージ本体を表示することができます。

webmail.cgi はパスワードを直接記述するといった大きな問題を抱えていますが、ちょっと IMAP の実験をしてみたいときなどには有用だと思います。

おわりに

IMAP を使ったのは今回が初めてですが、じつに〝使える〟ことが分かって嬉しいかぎりです。さきほども述べたように IMAP の仕様はかなり複雑で、これに完璧に対応した MUA を作るのはかなり大変そうです。しかし、とりあえず使ってみるといった程度なら、今回のようなプログラムでもなんとかなります。

POP+MH から IMAP+Web メールの環境にいきなり移動して大丈夫だろうかと心配でしたが、今回紹介したようなツールのおかげで、なんとか暮らしていけるメドが立ちました。今後、Wiki などから IMAP を活用する方法も考えたいと思っています。

現時点では、IMAP に対応している ISP は少数派ですが、POP よりも利用範囲が格段に広いので、今後の普及に期待したいところです。

今回紹介したプログラムはさきほど記した私のサイトで公開していますので、ご利用ください。

（ますい・としゆき　産業技術総合研究所）

[参考文献]

[1] Dianna Mullet、Kevin Mullet 著、木田直子訳『IMAP』、オライリー・ジャパン、2001 年

[2] T. W. Malone, K. R. Grant and F. A. Turbak, “The Information Lens: an intelligent system for information sharing in organizations”, in *Proceedings of the SIGCHI conference on Human factors in computing systems* (CHI86), pp.1-8, April 1986

リスト 2　webmail.cgi

```
#!/usr/bin/env ruby

require 'net/imap'
require 'net/smtp'
require 'cgi'
require 'kconv'

module Net
  class IMAP
    def fetch_attr(seq,s)
      fetch(seq,s)[0].attr[s]
    end
  end
end

class WebMail
  @@cgiurl = File.basename($0)

  def initialize(server,user,password)
    @server = server
    @user = user
    @password = password

    @cgi = CGI.new("html3")
    @cmd = @cgi['cmd'].to_s
    @mailbox = @cgi['mailbox'].to_s
    @mailbox = "INBOX" if @mailbox == ''
    @start = @cgi['start'].to_s.to_i
    @message_id = @cgi['message_id'].to_s
    @seq = @cgi['seq'].to_s.to_i
  end

  def init_imap
    @imap = Net::IMAP.new(@server,143)
    @imap.authenticate('CRAM-MD5',@user,@password)
    # @imap.login(userConfig, passConfig)
    @imap.examine(@mailbox)
  end

  def list
    init_imap
    messages = @imap.status(@mailbox, ["MESSAGES"])["MESSAGES"]
    (1..messages).to_a.reverse[@start..@start+9].collect { |seq|
      subject = @imap.fetch_attr(seq,"BODY[HEADER.FIELDS (SUBJECT)]").gsub(/Subject:\s+(.*)/,'\1')
      subject = "(no title)" if subject == ''
      from = @imap.fetch_attr(seq,"BODY[HEADER.FIELDS (FROM)]").gsub(/From:\s+(.*)/,'\1').⇒
             gsub(/.*<(.*)>.*/,'\1')
      message_id= @imap.fetch_attr(seq,"ENVELOPE").message_id
      read_cgi = "#{@@cgiurl}?mailbox=#{@mailbo}&message_id=#{message_i}&cmd=read"
      @cgi.a('href' => read_cgi){ subject} + "(#{fro})" + @cgi.br
   }.join +
      (@start >= 10 ?
       "<a href=#{@@cgiur}?mailbox=#{@mailbo}&start=#{@start-1}>前 </a> " : "") +
      "<a href=#{@@cgiur}?mailbox=#{@mailbo}&start=#{@start+1}>次</a>"
  end

  def read
    init_imap
    seq = @imap.search(["HEADER","MESSAGE-ID",@message_id])[0]
    from = @imap.fetch_attr(seq,"BODY[HEADER.FIELDS (FROM)]").gsub(/</,"&lt;")
    date = @imap.fetch_attr(seq,"BODY[HEADER.FIELDS (DATE)]")
```

```
    write_cgi = "#{@@cgiur}?mailbox=#{@mailbo}&seq=#{se}&cmd=write"
    date + @cgi.br + @cgi.a('href' => write_cgi){ from} + @cgi.br + @cgi.hr +
      @imap.fetch_attr(seq, "BODY[1.TEXT]").collect {|line| line + @cgi.b}.join
  end

  def write
    init_imap
    to = @imap.fetch_attr(@seq,"BODY[HEADER.FIELDS (FROM)]").gsub(/From:\s+(.*)/,'\1')
    from = "#{@use}@#{@serve}"
    subject = "Re: " + @imap.fetch_attr(@seq,"BODY[HEADER.FIELDS (SUBJECT)]").⇒
              gsub(/Subject:\s+(.*)/,'\1')

    text = (@seq == 0 ? '' :
     @imap.fetch_attr(@seq,"BODY[1.TEXT]").collect {|line| "> " + lin}.join)
    @cgi.form('method' => 'post', 'action' => "#{@@cgiur}"){
      @cgi.input('type' => 'hidden', 'name' => 'cmd', 'value' => 'deliver') +
'To: ' + @cgi.input('type' => 'text', 'size' => '40', 'name' => 'to', 'value' => to) ⇒
           + @cgi.br +
'From: ' + @cgi.input('type' => 'text', 'size' => '40', 'name' => 'from', 'value' => from) ⇒
           + @cgi.br +
'Subject: ' + @cgi.input('type' => 'text', 'size' => '40', 'name' => 'subject', ⇒
           'value' => subject) + @cgi.br +
@cgi.textarea('name' => 'text', 'rows' => '20', 'cols' => '60', 'wrap' => 'off'){ text} ⇒
           + @cgi.br +
@cgi.submit('value' => 'send')
    }
  end

  def mime_b (s)
    s.tojis.gsub(/\033\$B([^\033]*)\033\(B/){ |m|
      '=?iso-2022-jp?B?' + [m].pack('m').chomp + '?='
   }
  end

  def deliver
    from = mime_b(@cgi['from'].to_s)
    to = mime_b(@cgi['to'].to_s)
    Net::SMTP.start(@server, 25) {|smtp|
      body =  "To: #{t}\n"
      body << "From: #{fro}\n"
      body << "Subject: #{mime_b(@cgi['subject'].to_s}\n"
      body << "X-Mailer: Compact WebMail by masui@example.com\n\n"
      body << "#{@cgi['text'].to_s.toji}"
      smtp.send_mail(body,from,to)
   }
    @cgi.meta('HTTP-EQUIV' => 'refresh', 'CONTENT' => "1;URL=#{@@cgiur}")
  end

  def do_command
    cmd = @cmd.intern # write, read 等のコマンド判定
    body = (self.respond_to?(cmd) ? self.send(cmd) : list)
    @cgi.out {
      @cgi.html {
 @cgi.head { @cgi.title { "WebMail (#{@cm})"}} +
   @cgi.body { body.tosjis}
     }
   }
  end
end

webmail = WebMail.new('example.com','masui','mypasswd')
webmail.do_command
```