インターフェイスの街角 (52)— メールで情報検索

増井 俊之

どこでも検索

これまでに、個人用の情報管理ツールを何回かとりあげました。現在、私は Web については 2001 年 4 月号で紹介した Wiki Wiki Web を、PC 上では 2000 年 5 月号で紹介した Q-Pocket を、そして Palm 上では 2001 年 5 月号で紹介した Q-Pocket for Palm を使い、それぞれのデータを同期させながら情報を管理しています。これらのシステムの利用により、異なるシステム上で同じデータの検索・入力が可能になり、なかなか快適です。しかし、PC や携帯端末を持ち歩かなければ情報管理ができないという問題は残っています。

かなり軽くなってはきましたが、ノート PC はまだまだ重いので私はふだんは持ち歩いていません。PDA も肌身離さず持っているわけではないので、これらが手近にないときはまったく検索できないことになります。しかし、携帯電話はつねに持ち歩いています。最近は、PDA はともかく携帯電話はほとんどの人が持ち歩いているようですから、携帯電話で個人の情報検索ができれば便利です。

携帯電話で情報検索というと大層な感じがしますが、モバイル環境で情報が欲しくなることはよくあります。私自身の経験でも、次のような場合に検索ができればなぁと思ったことがあります。

- (初めての土地を訪れたとき) 友人がこの土地に引っ越したと言っていたのを思い出した。たしか、昔のメールに住所が書いてあったはずだが …… 。
- (あまり知らない土地で飲み屋を探すとき) 以前、知人に紹介してもらった飲み屋の場所を忘れてしまった。電話番号をファイルに書いた記憶があるが …… 。
- 訪ねようと思っているレストランのリストを PC で管理しているが、肝腎なときにリストを持ってくるのを忘れてしまった。
- (学会などで) 似たようなアイデアを聴いた記憶はあるが、詳しい内容が思い出せない。文献リストには書いてあるはずだが …… 。
- メールで割引セールの案内をもらったが、内容を忘れてしまった。メールが見られれば分かるのだが …… 。

私はこういう経験をすることが多いので、自宅やオフィスの PC に入っているメールやメモを自由に検索できれば、かなり役に立ちそうです。

事実、携帯電話で PC を操作する試みはいくつかおこなわれています。たとえば、Rajicon [1] は、i モード携帯電話の Java プログラムを使い、携帯電話のボタンで PC を操作しようというものです。携帯電話による PC の操作というのは画期的なアイデアですが、小さな画面とテンキーだけで PC を操作するのは容易ではなく、検索さえできればいい場合には最適な方法とはいえないでしょう。また、携帯電話と PC とのあいだで直接通信をおこなう必要があるので、固定 IP アドレスでインターネットに接続された PC しか操作できません。現在のところ、固定 IP アドレスの割当てを受けている家庭はわずかしかありませんから、この方法で自宅の PC の情報を検索したり操作できるのはごく一部の人に限られてしまうでしょう。

今後、情報家電製品が普及していくと、家庭の情報機器やセンサーを外部から操作したいという要望が増えそうです。しかし、どこにいても自宅に直接接続できる環境が一般化するまでにはまだまだ時間がかかりそうなので、とりあえずは別の方法を考えなければなりません。

昔のいわゆる〝ホーム・オートメーション〟機器を外部から制御する場合は、電話線を利用するのが普通でした。現在も、固定電話の回線を用いて家庭の機器を制御するシステムは多いようです。たとえば、留守中に電話経由でペットに餌をやるシステムなどが商品化されています。

家庭内ネットワークに直接アクセスする場合には、ネットワークへのアクセスだけに使う電話を家庭に用意する方法が考えられます。すぐに思いつくのは、固定電話の回線にモデムを設置し、外部からの着信を受け付ける方法です。しかし、せっかくブロードバンド接続された家庭内ネットワークでアナログモデムによる着信というのではあまり恰好がよくありません。家庭内ネットワークに外部から呼び出せる携帯電話を付け、これによって家電機器の操作を実現するシステムもあります。たしかに、どこからでも家庭内ネットワークに直接接続できるのは便利です。とはいっても、そのためだけに専用の電話回線を契約してもよいと考える人は少ないのではないでしょうか。

このように、現時点では、一般家庭の PC などに外部から手軽に直接アクセスするためのこれといった方法はないのが実情です。

メールによる情報検索

最近、ADSL などによる常時接続環境が急速に普及しつつあります。このような接続形態では、PC 側から Web やメールにアクセスするのはたいへん簡単です。PC を〝御用聞きモード〟にしておき、外部からの検索などの要求を定期的にチェックする仕組みにすれば、家庭の PC に要求を送ることができます。定期的に状態をチェックすることを〝ポーリング (polling)〟といいます。さきほど挙げた例のような情報検索要求は一刻を争うものではないので、ポーリングによる要求処理で十分だと思います。

家庭の PC でポーリングされる検索要求を保持するには、メールサーバーが使えます。まず、PC からメールサーバーに対して定期的に POP で接続し、スプールされているメールを覗いて検索要求の有無を調べます。そして、要求があったら検索を実行し、その結果を問い合わせてきたメールアドレスに返送すればいいでしょう。今回は、このようなポーリングにもとづく検索システムを、ちょっと宗旨変えをして Perl ではなく Ruby で作成してみました。

図 1　メインプログラム

```
#!/usr/bin/env ruby
# 'pimagent' program

require 'pimagent.rb'

PimConfig = {
  :datadir => '/cygwin/home/masui/PIM',
}

MailConfig = {
  :smtpserver => 'smtp.csl.sony.co.jp',
  :popserver =>  'pop.csl.sony.co.jp',
  :user =>       'masui',
  :password =>   'abcdefg',
  :from =>       'masui@csl.sony.co.jp',
  :to =>         'masui@csl.sony.co.jp'
}

pa = PimAgent.new(MailConfig, PimConfig)

loop do
  begin
    pa.do_search
  rescue
    puts "connect fail"
  end
  sleep(60)
end
```

メインプログラム

図 1 がメインプログラムです。検索をおこなうエージェント PimAgent は pimagent.rb で定義し、検索メソッド do_search を適当な時間間隔で呼び出しています。

検索対象データは PimConfig に、メール関係の情報は MailConfig に設定しておきます。

初期化とメインルーチン

PimAgent クラスの実体は、図 2 の pimagent.rb で定義します。

検索メソッド do_search では、次のような処理をおこないます。まず、POP サーバーに接続して each_mail メソッドでメールを 1 通ずつ取得します。そして、メールに検索要求文字列が含まれていれば検索パターンを取り出し、検索した結果をメールで通知します。

メールの読出し (POP) と送出 (SMTP)

図 3 のメソッド each_mail は、POP サーバーに接続

図 2　pimagent.rb

```
# pimagent.rb
class PimAgent
  def initialize (config = {}, pimconfig = {})
    @smtpserver = (config[:smtpserver] or raise ArgumentError)
    @popserver = (config[:popserver] or raise ArgumentError)
    @user = (config[:user] or raise ArgumentError)
    @password = (config[:password] or raise ArgumentError)
    @from = (config[:from] or raise ArgumentError)
    @to = (config[:to] or raise ArgumentError)
    @header = {}
    @performed = {}
    @pimconfig = pimconfig
  end

  def do_search
    each_mail {|mail|                  # POPサーバー上のすべてのメールを調査
      if search_requested?(mail) &&    # 検索要求があり、かつ
         !search_performed?(mail)      # まだ検索していなければ
        pattern = get_pattern(mail)    # 検索パターンを取得して
        result = search(pattern)       # 検索を実行し、
        send_mail(result) if result    # 結果をメールで返す
      end
    }
  end
end
```

図 3　メソッド each_mail

```
class PimAgent
  require 'net/pop'
  def each_mail
    Net::POP.start(@popserver,110,
                   @user, @password) { |pop|
      puts 'connect'
      pop.each { |mail|
        yield mail
      }
    }
  end

  require 'net/smtp'
  require 'kconv'
  def send_mail (txt)
    Net::SMTP.start(@smtpserver, 25) do |smtp|
      smtp.send_mail(txt.tojis, @from, @to)
    end
  end
end
```

してから 1 通ずつメールを処理するイテレータで、POP クラスの each メソッドをそのまま利用しています。

データ検索

データの検索には、図 4 の PimDatabase クラスを使用します。これは、2000 年 5 月号で紹介したデータベース検索の手法を Ruby で実装したものです。

図 4　PimDatabase クラス

```
class PimAgent
  require 'pimdb'     # PimDatabaseクラス
  def search (pat)
    result = nil
    re = Regexp.new(pat)
    pim = PimDatabase.new(@pimconfig)
    pim.each { |id|
      txt = (pim.text(id) or "")
      title = (pim.title(id) or "")
      if re.match(txt) || re.match(title)
        result = title + "\n\n" + txt
        puts 'found'
        break
      end
    }
    pim.close
    result
  end
end
```

ヘッダ取得と解析

今回は、検索コマンドは Subject:フィールドで指定することにします。たとえば、

`Subject: search` *keyword*

のような行がヘッダに含まれていたら、*keyword* を検索キーワードとみなします。同じキーワードで何度も検索す

図 5　ヘッダの取得と検索キーワードの取得

```
class PimAgent
  require 'mime'
  def search_requested? (mail)
    get_header(mail)
    subject = @header[mail]['Subject']
    if subject
      subject = subject.decode64.toeuc
    end
    subject && subject =~ /^search\s+/
  end

  def get_header(mail)
    if !@header[mail]
      lines = mail.header.split(/[\r\n]+/)
      data = {}
      until lines.empty?
        line = lines.shift
        break if line.empty?
        if /^(\S+):\s*(.*)/=~ line
          (attr = $1).capitalize!
          data[attr] = $2
        else
          line.gsub!(/^\s*/,'')
          data[attr] += ("\n" + line)
        end
      end
      @header[mail] = data
    end
  end
  def search_performed? (mail)
    get_header(mail)
    msgid = @header[mail]['Message-id']
    performed = @performed[msgid]
    @performed[msgid] = 1
    performed
  end

  def get_pattern (mail)
    get_header(mail)
    subject = @header[mail]['Subject']
    subject = Kconv::toeuc(subject.decode64)
    subject =~ /^search\s+(.*)$/
    $1
  end
end
```

図 6　検索指示メールの例

```
To: masui@csl.sony.co.jp
From: masui@csl.sony.co.jp
Subject: search 買い物
```

ることを防ぐために、search_performed メソッドを使用しています（図 5）。

検索例

このプログラムを常時動かしておけば、図 6 のようなメールを送ることで Subject:行に含まれるキーワードが検索され、図 7 のメールが返送されます（これは、秋葉原に行ったとき買おうと思っている小物のリストです）。

おわりに

特殊な設定をいっさいおこなわず、メールで家庭の PC に蓄えたデータの検索ができるのはたいへん便利です。私は喜んで使っているのですが、周囲の評判は残念ながらいまひとつといったところです。これには、以下のような理由があるようです。

- モバイル環境で検索などしたいとは思わない。
- そもそも、PC でデータベースを管理していない。

図 7　検索結果として返ってきたメール

```
Date: Thu, 21 Feb 2002 15:23:25 JST
From: Toshiyuki Masui <masui@csl.sony.co.jp>
Message-Id:  <200202210623.PAA94051@hotaka.csl
.sony.co.jp>

買い物リスト

Cross ion
USBマウス
モデムセーバ
太陽電池
……
```

- ふだんから、PC の電源は切るようにしている。
- 単純なキーワードでは必要なデータがみつからない。

とはいえ、すくなくとも私にとっては役に立っているのは事実です。先日も、このシステムのおかげでパスポート番号を調べることができましたし、買い物リストのチェックにもよく使っています。こんなときには便利、という実例を積み重ねて普及を目指したいと考えています。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Norman Su, Masahiko Tsukamoto and Shojiro Nishio, *Rajicon: A System for Remote PC Access through a Cellular Phone*, 情報処理学会シンポジウム・シリーズ（マルチメディア，分散，協調とモバイルシンポジウム）, Vol.2001, No.7, pp.349–354, June 2001