TSNET スクリプト通信

TSC 編集委員会

ISSN 1884-2798

目次

表紙写真: あーかい、さくらんぼ
撮影: jscripter
日時: 2012 年 5 月 20 日
場所: 宇品波止場公園(広島市)
メモ:
Wikipedia によると、サクラも 400 種もあるとかで、欧米では実を食べるための改良が行われ、日本では花の改良が行われている。突然変異も多いとか・・・サクランボは、その名の通り、サクラの実であるということを、加古川からのツイッター返信で再認識したというお話。

巻頭言

jscripter

広島も梅雨入り、暑くなった。もちろん、夏だから当然だ。五月末に投稿していただいた二件の記事を抱えたまま、六月に入った。自分自身の記事が形を成していないからだ。

欧州債務危機は解決の兆しはなく、暗雲の中に漂流している。米国のシェールガス革命は回りまわって、日本のガソリンの価格を押し下げはじめた。来週からは、アップルの WWDC2012 が始まる。すべての要素は社会の坩堝の中でかき回され相互作用するはずだが、何が起こるのだろうか。

今号では、機械伯爵氏の「Python の文法」草稿　第 7 回、バイト列、バイト配列関連のクラス・メソッドが登場。ムムリクさんは、「 HTA でお手軽(?)GUI」で ActiveScriptRuby で HTA(HTML Application)を作られている。Windows 上であれば、アプリケーション作成の一つの選択肢になる。

僕はと言えば、DesktopWeb フレームワークの展開をどうすべきかで足踏み状態。単純な DDL(Diary Description Language)→HTML→RSS/Atom 変換をさらに進化させて、日記の意味データ化を推し進めようという構想なのだが、記述形式に何を選択するかというところで、いろいろと漁っている。なかなかよいものがない。自然言語に近い人間可読形式で、データとしての取り扱い要素を内在化しているものという過大な要求をするので無理からぬところだろう。今回は、Compact Topic Maps を取り上げて、使い道を探ってみよう。

最近、Facebook も熱心に使ってみたのだが、Twitter も含めて情報の記録に使うのは邪道だという結論に至っている。なぜなら、SNS はその名の通り、社会的な場でもあるからだ。情報の記録はやはり Blog(Weblog)に集中させて考えるべき。それでなおさら、DesktopWeb が大切という方向性が浮かび上がる。これを我田引水とも言う。

（投稿: 2012 年 6 月 9 日）

Python の文法

TSNET スクリプト通信版　草稿　第７回

6-2-2-2-3-2-2 バイト列を扱う順序列型

バイト列(bytes)は、1byte=8bit の情報を一つのアイテムとして扱う、一種の順序列型です。
歴史的にはPython2 までは、このバイト列を扱うコレクションとして'str'型が使用されてきました。
しかしPython3 では、'str'クラスは完全に「（マルチバイトを含む）文字」を扱うコレクションになり、替わりに'bytes'(バイト列)クラスと'bytearray'(バイト配列)クラスの二つの型がバイト列操作専用クラスとして登場しました。
バイト列型のメソッドは、大部分が文字列と同じで、一部バイトオブジェクト操作用のメソッドが加えられ、不要なメソッドが削除されています。
バイト配列型は可変型のバイト列で、バイト列のメソッドに汎用可変順序列型のリスト('list')クラスと似たようなメソッドがいくつか追加されています。
以下に、文字列、バイト列、バイト配列、リストのメソッド一覧を載せます（全てが持つメソッド及び文字列あるいはリストのみが持つメソッドは省略してあります）

【メソッド一覧】

[method]	str	bytes	list	bytearray
__alloc__				○
__delitem__			○	○
__getnewargs__	○	○		
__iadd__			○	○
__imul__			○	○
__setitem__			○	○
append			○	○
capitalize	○	○		○
center	○	○		○
decode		○		○
endswith	○	○		○
expandtabs	○	○		○
extend			○	○
find	○	○		○
fromhex		○		○
index	○	○	○	○
insert			○	○
isalnum	○	○		○
isalpha	○	○		○

[method]	str	bytes	list	bytearray
isdigit	○	○		○
islower	○	○		○
isspace	○	○		○
istitle	○	○		○
isupper	○	○		○
join	○	○		○
ljust	○	○		○
lower	○	○		○
lstrip	○	○		○
maketrans	○	○		○
partition	○	○		○
pop			○	○
remove			○	○
replace	○	○		○
reverse			○	○
rfind	○	○		○
rindex	○	○		○
rjust	○	○		○
rpartition	○	○		○
rsplit	○	○		○
rstrip	○	○		○
split	○	○		○
splitlines	○	○		○
startswith	○	○		○
strip	○	○		○
swapcase	○	○		○
title	○	○		○
translate	○	○		○
upper	○	○		○
zfill	○	○		○

バイト列、バイト配列の項では、特有のメソッドの説明のみ載せます。それ以外は、文字列／リストのメソッドの項を参照してください（なお、バイト列／バイト配列のみに共通するメソッドは、バイト列の項で説明します）

6-2-2-2-3-2-2-1 バイト列(bytes)

■クラス名：'bytes'

１バイト情報のシーケンスを扱うクラスです。(ASCII)文字列の前に'b'を加えたリテラルで記述（ASCII 文字以外は、¥x をつけた 16 鍼法表記）する他、'bytes'クラスに以下のようなオブジェクトを与えてもバイト列は生成できます。

・0～255 の整数がアイテムであるリスト、タプル、反復子
・バイト列、及びバイト配列オブジェクト
さらに、文字列を引数無しの'encode'メソッドでエンコードしても、バイト列は生成できます。

```
>>> b0 = b'abcde'
>>> b0
b'abcde'
>>> b1 = bytes([97,98,99,100,101])
>>> b1
b'abcde'
>>> b2 = 'abcde'.encode()
>>> b2
b'abcde'
```

なおコンソールへの出力は、ASCII コードが対応している文字がある場合はその文字で、なければ 16 進法表示で表されます。

```
>>> bytes([124,125,126,127,128])
b'|}~\x7f\x80'
```

ただし、文字列を bytes クラスに直接渡しても、バイト列は出来ません。

```
>>> bytes("abcde")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string argument without an encoding
```

※Unicode の UTF-8 エンコーディングは、ASCII コード互換です。

実際の性質としては文字列というより、0～255 の数値をアイテムとするタプルのような性格のオブジェクトです。

```
>>> b = bytes((x for x in range(10)))
>>> b[0]
0
>>> b[1]
1
>>> b[2]
2
```

ただし、文字列と共通するメソッドは数多く実装されています。

以下に説明するメソッドは、バイト列（及びバイト配列）特有のメソッドです（文字列、リストと共通するメソッドについては、リスト及び文字列の記述を参照してください。なお、「バイト列を扱う順序列型」の項に共通メソッドの一覧を挙げてあります）

■文字列に戻す

書式：b.decode() ⇒ s

動作：バイト列"b"を文字列"s"に戻す。ただし、UTF-8 エンコードで解釈できないコードの場合はエラーが返る

■16 進法表示から、バイト列を作成する

書式：bytes.fromhex(S)

動作：文字列"S"の先頭から数字(0～F)を２文字ずつセットで読み取り、16 進法と解釈してバイト列を生成する

※'fromhex'で読み取る文字列の間にホワイトスペースがあっても構いませんが、２つペアでないとエラーを返します。
つまり、
bytes.fromhex("FFFF FF FF")
は通りますが、
bytes.fromhex("FFFF F F FF")
は通りません

```
>>> s = "abcde"
>>> b = s.encode()
>>> b
b'abcde'
>>> list(s)
['a', 'b', 'c', 'd', 'e']
>>> list(b) # 似てるが文字列ではない
[97, 98, 99, 100, 101]
>>> [hex(x) for x in b]
['0x61', '0x62', '0x63', '0x64', '0x65']
>>> bytes.fromhex("6162 636465") # 16進法で生成
b'abcde'
>>> bytes.fromhex("616 263 6465") # 2桁ずつになっていないと・・・
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: non-hexadecimal number found in fromhex() arg at position 2
>>> ub = b"ぱいそん" # ASCII文字でないと、リテラルで書けない
  File "<stdin>", line 1
SyntaxError: bytes can only contain ASCII literal characters.
>>> ub = "ぱいそん".encode() # encodeを使えばよい
>>> ub
b'\xe3\x81\xb1\xe3\x81\x84\xe3\x81\x9d\xe3\x82\x93'
>>> ub.decode()
'ぱいそん'
>>> ub2=bytes.fromhex("e381b1e38184e3819de38293")
>>> ub2.decode()
'ぱいそん'
```

6-2-2-2-3-2-2-2 バイト配列(bytearray)

■クラス名：'bytearray'

可変長のバイト配列を扱うクラスです。バイト列を渡してつくる以外は、バイト列クラスが受け取ることの出来るタイプのオブジェクトで生成することが出来ます（無意味ですが、'bytes'クラスにバイト列オブジェクトを渡してもバイト列を生成できますので、'bytes'クラスと'bytearray'クラスの受け取ることの出来る引数は全く同じです）

なお、バイト配列オブジェクトのリテラル及び表示は

```
bytearray(b'abcde')
```

のような形になります。

バイト配列のメソッドは、バイト列及びリストと殆ど共通です（バイト列にリストのメソッドを一部加えたものがバイト配列のメソッド群となっています。詳しくは「バイト列を扱う順序列型」の項を参照）。

独自のメソッドは以下に挙げる'__alloc__'一つです。

■割り当てられているメモリ容量

書式：ba.__alloc__() ⇒ i

動作：バイト配列オブジェクト"ba"に現在割り当てられているメモリ容量を整数"i"として返す

※バイト配列のアイテムを増減してみると、割り当てられているメモリ容量は必ずしもアイテム数にきっちりと比例しているわけでないのが分かります。データとして保存するならば、バイト列オブジェクトに変換したほうが、メモリ効率が良いことは言うまでもありません

【Tea Break】先行き予測の難しさ ～Python3への移行の現状～

「Pythonの文法」では、Python3(3.0以降)の文法に限って紹介していますが、巷ではなお、Python2系であるPython2.7（Python2系では最後になるとアナウンスされながら）が依然として勢力を保っています（卑近な話で言えば、去年末からandroidスマホに替えた私は、SL4AのPythonが2.7だと知って愕然としました）

その中で、Python3最新のver.3.3に加えられた変更のうちPEP414の「Unicode文字は、また頭にuを付けても良い」という変更は、ある意味でPython3側からの譲歩です。とにかく２系から移行してもらわないことには、Pythonのスタイルが恒久的に分裂してしまうかもしれないという危機感の表れかもしれません。

これは、2008年末にPython3が発表された頃には、私が予想もしていなかった事態です。

Python3の文法にまずまず満足していた(reduceくらい仕方ないか、lambdaは残ったし)私は、Python2から3への移行は、遅くとも２，３年で完成すると思っていたのです。

2012年、あれから３年以上経ちましたが、Python2に完全移行する兆しは見えてきません（そろそろ移行のスピードが速くなる、という予測は出ていますが、実感は微妙です）

やはり静的コンパイル言語と異なり、動的にコンパイル可能な言語の文法は（ソースコードレベルで残っている分）おいそれとは後方互換を捨てられないようです。

何の発言力の無い私としては、Python3の魅力を地道に紹介していく他に方法はありません。

オブジェクト指向言語としてのスタイルが格段に安定し、また、関数型プログラミングのパラダイムも導入しやすくなったPython3は、従来の寄せ集め風ののPython2系より綺麗で「使いやすい」言語の筈です。Pythonプログラマの方々が、慣れ親しんだスタイルを捨てがたくてPython3に変えないとは思えません。プログラミングスタイルの変更は非常にわずかですし、反復子の問題など今までより面倒になることは多少はありますが、楽になる部分はそれ以上に大きいと思います。多分、今までの遺産の問題なのでしょう（それだけに「新しく実装された」筈のSL4Aが2.7系なのは解せません。何か根本的に問題があるのだろうか？）

せめて今から移植される実装系には、Python3が基本となってもらえるとありがたいです。

（投稿：2012年5月25日）

HTA でお手軽(?)GUI

ムムリク

ReVIEW というプログラムがあります。ここ数年勃興してきた電子書籍の何度目かというブームもあって昨年あたりから人気が高まっているツールのひとつです。もともと出版原稿を執筆するために作られてきた経緯があるようですが、PDF や EPUB の生成も行うことができるように拡張されてきました。

基本的に原稿はテキストファイルとして用意されます。そこに見出しであるとか、各種リストであるとか脚注であるとか、さまざまな注記を独自の記法で追加し、基準となる原稿テキストファイルを作成します。

このテキストファイルがあれば、あとはどのような形に仕上げたいのかを指定するだけで望みの形のファイルが生成されます。

しかし、これらのタグつけを正しく行っていくのはなかなか面倒です。ブログの編集画面のようにいろいろのタグが簡単に設定できたら楽ではなかろうか、ということでプログラムを作ってみました。

HTA (HTML Application) なので基本的に Windows 限定です。実行には ASR (Active Script Ruby) が必要です。

HTA は基本的にはウェブページの HTML と同じです。ボタンやテキスト表示の枠などを好みにレイアウトできます。

それぞれのボタンなどを押したときの処理を好みのスクリプト言語で記述できますが、ここでは Ruby で記述します。手軽に GUI なプログラムを作れるという点でもなかなか便利です。

■必要なもの

Ruby 1.9.x (http://www.artonx.org/data/asr/)

ASR が必要なのでこちらのバイナリパッケージをインストールしてください。

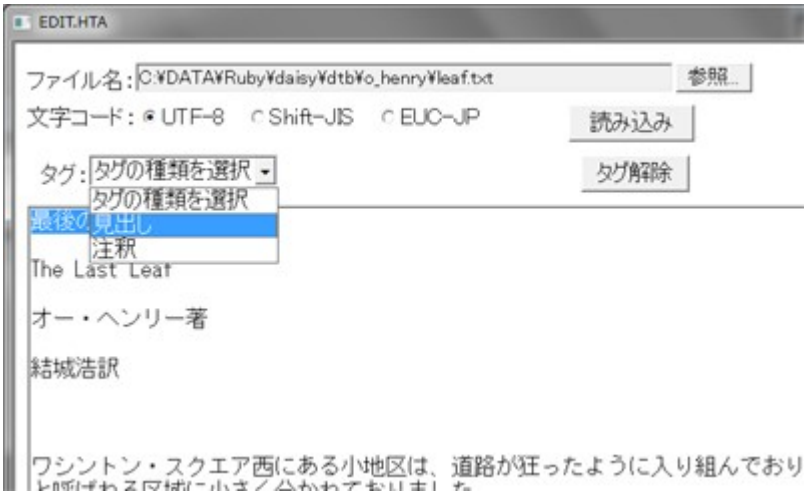
■使い方

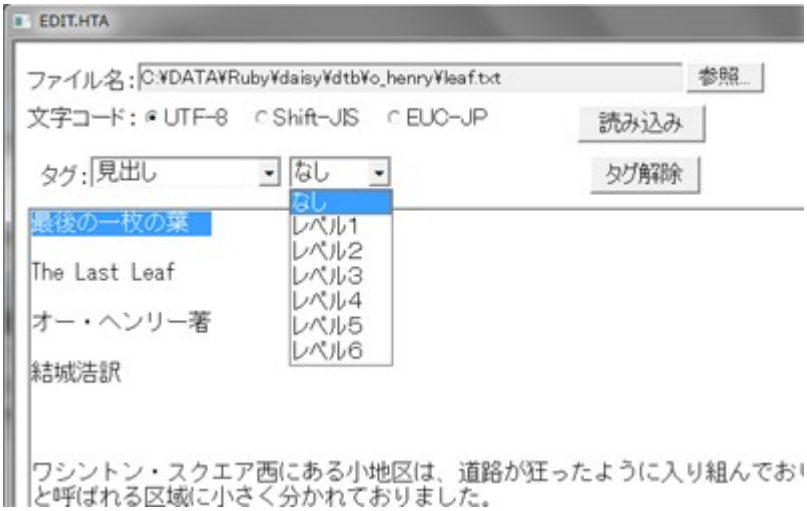
ここではごく簡単なものからはじめてみます。機能としてはこんな感じです。

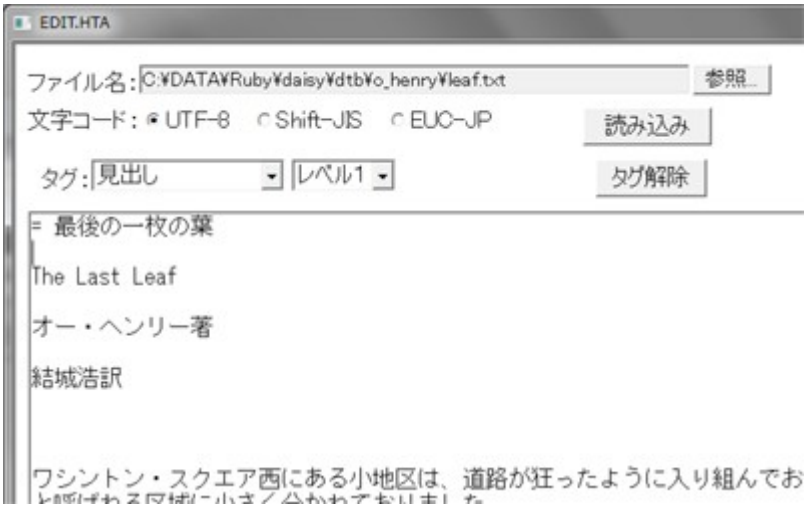
１．ファイルを指定して開く。
２．記法に沿った専用タグをつける。
３．ファイルを保存する。

edit.hta のアイコンをダブルクリックして実行します。Window が開きます。「参照」ボタンを押して編集するテキストファイルを選択し、該当する文字コードをチェックして、「読み込み」ボタンを押します。テキストが表示されます。

タグ付けをしたいテキストの部分をマウスで選択します。選択された状態でタグの種類を選び、さらに表示される個別のタグを選択すると、テキストにタグが追加されます。







タグを解除する場合にはタグを含む部分をマウスで選択し、「タグ解除」ボタンを押します。

編集が完了したら「保存」ボタンを押します。ファイル名は元のファイル名に -edit を付けたものになります。（sample.txt であれば、sample-edit.txt）

■プログラムについて

全体は edit.hta を見てください。冒頭の head 部分 script 内に Ruby のプログラムを書きます。続く body 内には通常の HTML 同様にウェブページのイメージで HTML タグを書きます。なお、このプログラムでは ReVIEW とはやや違うタグ仕様です。

ファイルインプットなどそれぞれの部分には異なる id を設定しておきます。

ファイル名の取得は、

```
<input type="file" id="path" size=60>
```

です。

```
<input type="button" id="go" value="読み込み" onclick="load">
```

こちらは読み込みボタン。ボタンがクリックされると、onclick で指定された load に処理が移ります。ボタンや入力枠などのパーツそれぞれへのアクセスは、

```
@window.document.all("id名")
```

もしくは、

```
@window.id名
```

でいけるようです。その示す値は、value で取り出せます。

load では、パスが指定されていればそのファイルを開いてすべて読み込み、それを表示の

textarea の値に代入して表示させています。

文字コードのラジオボタンは def encode; end 部分で処理します。ラジオボタンの値は配列で返されるので、チェックされているかどうかを見て、チェックされている値を @encode に代入します。

テキストエリアに表示する際に文字列にたいして encode("Windows-31J") としていますが、これは現在のところ ASR の仕様上 UTF-8 などに対応していないためです。よって UTF-8 などに固有の文字を使用している場合には正しく処理することができません。これは保存に関しても同じです。

def tags; end はタグの大きな種類分けのリストを表示する部分です。ここで選択した種類に応じて詳細なリストを のなかに表示させます。

その詳細なタグのリストに応じた処理が、def level; end や def note; end です。level では見出しを、note では脚注を処理します。

タグ解除ボタンが押されたときには、def off; end が呼ばれます。実際に解除しているのは def tagoff; end 部分です。

textarea のなかの選択された文字列を取得しているのは、

```
s = @window.editarea.document.selection.createRange()
```

の部分です。

基本的な処理はこれだけです。あとはそれぞれのタグごとのリストとそれぞれにそった処理を施す部分を作ることでさまざまなタグを簡単に間違いなく設定・解除することができるようになります。

■UTF-8 への暫定対応

どうしても UTF-8 を使わざるを得ないときには、エディタなどでファイルを開いておいてから全体をコピーして、直接編集用のエリアに貼り付けます。保存の時には逆に編集エリアのテキスト全体をコピーしてエディタなどに貼り付けてからエディタ側で保存します。

■まとめ

HTA で作るうえでの悩みは情報があまりないということかもしれません。今回もずいぶんと検索などして調べたのですが、選択文字列情報などをどう取得するのかになかなか出会えませんでした。全般的に情報は少なめです。マイクロソフトのサイトですら今ではあまり見つかりません。

レイアウトについても css で調整していくにはなかなか慣れが必要なようで、あまりスマートなものではないかもしれません。そのあたりをあまり気にしなければ、手軽につくれるというのはあると思います。

今となってはやや情報も少なめではありますが、ちょっとしたプログラムではあるけれど GUI にしたい。できるだけ追加でインストールなど不要なほうがよい。という時には案外使えるかもしれません。

[edit.hta]

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=Windows-31J" />
<title>EDIT.HTA</title>
<script language="RubyScript">
LEVEL_LIST = %Q[<select name="headlevel" onchange="level">
   <option value="0" selected>なし
   <option value="1">レベル１
   <option value="2">レベル２
   <option value="3">レベル３
   <option value="4">レベル４
   <option value="5">レベル５
   <option value="6">レベル６
</select>]
NOTE_LIST = %Q[<select name="note" onchange="note">
   <option value="0" selected>なし
   <option value="1">注釈番号
   <option value="2">注釈
   <option value="3">製作者注（ブロック）
   <option value="4">製作者注（インライン）
</select>]

def load
   encode()
   edit = @window.editarea
   @f = @window.document.all("path").value
   unless "" == @f
      lines = ""
      File.open(@f, "r:#{@encode}") {|file|
         file.each_line {|line|
            lines += line
         }
      }
      edit.value = lines.encode("Windows-31J")
   else
      @window.alert "ファイルを指定してください"
   end
end

def save
   lines = @window.editarea.value.split(/¥n/)
   extname = File.extname(@f)
   basename = File.basename(@f, extname)
   fname = "#{basename}-edit#{extname}"
   p = File.dirname(File.expand_path(@f))
```

```
   Dir.chdir(p)
   File.open(fname, "w") {|f|
      lines.each {|line|
         f.print line
      }
   }
end

def encode
   @encode = ""
   enc_chk = @window.document.all("encode")
   enc_chk.each {|enc| @encode = enc.value if enc.checked}
end

def tags
   tag_gloup = @window.document.all('tags').value
   span = @window.taglist
   case tag_gloup
   when '0'
      select = ""
   when '1'
      select = LEVEL_LIST
   when '2'
      select = NOTE_LIST
   end
   span.innerHTML = select
end

def level
   s = @window.editarea.document.selection.createRange()
   l = @window.document.all('headlevel').value
   unless "" == s.text
      if "0" == l
         tagoff(s)
      else
         level = "=" * l.to_i
         s.text = "#{level} #{s.text}"
      end
   else
      @window.alert "文字列が選択されていません"
   end
end

def note
   s = @window.editarea.document.selection.createRange()
   n = @window.document.all('note').value
   unless "" == s.text
      case n
      when '0'
```

```
            tagoff(s)
        when '1'
            s.text = "@<fn>{,#{s.text}}"
        when '2'
            s.text = "//footnote[]{¥n#{s.text}¥n//}"
        when '3'
            s.text = "//prodnote[o]{¥n#{s.text}¥n//}"
        when '4'
            s.text = "@<pn>[o]{#{s.text}}"
        end
    else
        @window.alert "文字列が選択されていません"
    end
end

def off
    s = @window.editarea.document.selection.createRange()
    unless "" == s.text
        tagoff(s)
    else
        @window.alert "文字列が選択されていません"
    end
end

def tagoff(select)
    t = select.text
    if /@<fn>{[^,]*,([^}]+)}/ =~ t
        t.sub!($&, $1)
    elsif /@<pn>¥[[or]¥]{([^}]+)}/ =~ t
        t.sub!($&, $1)
    else
        t.sub!(/=+¥s+/, "")
        t.gsub!(%r!¥A//[a-z][a-z]+.*(?:¥n|¥z)!, "")
        t.gsub!(%r!¥n//}¥z!, "")
    end
    select.text = t
end

</script>
<style type="text/css">
<!--
body {font-size: 1.1em;}
#path {font-size: 0.9em;}
#box {margin-top: 0.5em; width: 41em;}
#encodes {width: 25em;float: left;}
#load {width: 4em;float: left;}
#save {margin-left: 0.4em;width: 4em; float: left;}
#box2 {width: 42em;margin: 10 10 10 10;}
#select {width: 25em;float: left;}
```

```
#tagoff {width: 5em;float: left;}
#text {float: clear;}
#go, #save, #tagoff {font-size: 1em;}
textarea, select {font-size: 1em;}
-->
</style>
</head>
<body>
<div>
ファイル名：<input type="file" id="path" size=60>
</div>
<div id="box">
   <div id="encodes">
文字コード：<input type="radio" name="encode" value="UTF-8" checked>UTF-8
  <input type="radio" name="encode" value="Windows-31J">Shift-JIS
  <input type="radio" name="encode" value="EUC-JP">EUC-JP
   </div>
   <div id="load">
      <input type="button" id="go" value="読み込み" onclick="load">
   </div>
   <div id="save">
      <input type="button" id="save" value="保存" onclick="save">
   </div>
</div>

<div id="box2">
   <div id="select">
タグ：<select name="tags" onchange="tags">
         <option value="0" selected>タグの種類を選択
         <option value="1">見出し
         <option value="2">注釈
      </select>
      <span id="taglist"></span>
   </div>
   <div id="tagoff">
      <input type="button" id="tagoff" value="タグ解除" onclick="off">
   </div>
</div>

<div id="text">
   <textarea rows="20" cols="84" wrap="soft" id="editarea"></textarea>
</div>
</body>
</html>
```

（投稿：2012 年 5 月 26 日）

覚醒と進化 試論 III - Compact Topic Maps

jscripter

1. 更新日記 rewrite 計画

いよいよと言うか、いや、今更というか^^;)、待てども待てども(実は出てくることを心の片隅では恐れているのだが)僕の眼鏡に叶う究極のアプリは登場しないので、結局、自分で作るということになる。仕様を開示せよと言われそうだが、設計図は断片的であり、ある意味既に断片的に存在しているものであり、新しい要素は Web の発展とともに登場しつつある。

デスクトップには、文書や写真、音楽・エアチェックなどのメディアのファイル管理システムがローカルのウェブや iTunes につながっている。更新 日記の出版システムは Zed とローカルのウェブで動作している。その他、メモカレンダーと一緒にデスクトップにメモるシステムがあるがほとんど使わない。

定期的に確実に動いているのは、更新日記の出版システムであり、エアチェックの Podcast 配信サーバーである。その他、最近では、Twitter への日記のカテゴリとタイトルとリンクの配信を再開している。

Web の新しい要素としては、Evernote や Dropbox などのクラウドサービスがあるが、モバイル環境を使うことはほとんどないこととコストを含めて記憶容量に限界があることは明白なので、今のところ、ファイル共有が必要な時だけの限定的な利用に留まっている。Evernote へのノートの自動書き込みは最近試してみたところだが、これも Evernote の使用がメモ的なものに留まっている以上、今のところ注力する必然性に乏しい。

- 日記オブジェクトプロトコル入門(03/04/2012)

一方、家庭内で日常的に使用するコンピューティング・デバイスが増え、iPod touch/iPad から、Windows のデスクトップ(ローカル)サーバーにアクセスするというニーズが出てきた。DHCP による IP アドレスを取得する方法にはこれまで検討してきたように限界がある。昨年末に知った Pagekite(+ownCloud)を利用すれば面白いことに気づいていたのだが、Ubuntu のアップグレードがあったりして、そのままになっていた。

- Pagekite と ownCloud (2011/12/03)

話は少し変わるが、最近、柴田さんの gdrive-cli の記事を見て、Linux で動かしてみるかと思い立ち、次いでに eban さんの RSS で 知った Ubuntu の 12.04 のアップグレードを試みた。確かめてみると、前には標準ではインストールされていなかった ownCloud が 3 にアップ デートされていた。ブラウザから pagekite には繋がらなかったので、最初から、curl でインストールするところから始めた。アカウントは残っていたようだが、前のパスワードも記録できていなかったので、変更してトライ。当然のことながら、これを使えば、モバイルを含めて、様々なデバイスからアクセスできるようにすることも可能だ。実際、Windows 上からファイルをアップロードして、iPad などからアクセスできる。

pagekite は、個人利用(individuals)は無料か、自分の払いたいだけ払ってくださいということになっている。できれば$3/月 払ってくださいということ。本格的に利用するようになれば払っても良いかな。他の利用は$5.99/月。様々な個人用のクラウドサービスが増えているので、必要性がどこまであるかという感じはするが、自前のサーバーが準備できればいろいろと応用は可能だろう。

サーバー環境はともかく、表題のリライト計画。Topic Maps の概念を導入することによって、日記の意味的な構造化を図ろうというのが目的にある。サーバーの移転の問題もあるが、日記の表示も意味的な構造に依存することになる。

現在の記事の作成は Zed 上の手作り的な要素が多いのだが、一次データを元に自動化したいというニーズがある。もう一つはタグ付けによる分類の強化。記事の意味的な関連付けの新しい方法を考え

ること。そして、記事を関連付ける新しい記事の生成の方法などを考えている。手作り的な要素と記事を関連付ける仕組みを連動させる感じ。

回顧2011年、展望2012年 - アップル編の記事によれば、音楽配信の動向 - 新しいオーディオの世界はどうなるかに次のように書いている。

> ・・・iTunesのようなソフトウェアやiPodのようなハードウェアを使う音楽配信システムは、音楽データにメタ情報や関連情報等を付加して、 如何に実体のあるもののように見せることができるかが重要である。これも情報を大量に集積しつつ、全体と部分を見通しよく取り扱う問題に通ずる。いずれは、DVD/CDプレーヤ自体をコンピュータにつないで音楽やビデオをデータベース化し、情報をPCで閲覧しながらオーディオを制御したいものだと思う。

僕に必要な日記システムは、各種のコンテンツとiTunesやWebブラウザなどのアプリケーション、各種のサイトやWebサービスを統合するものだ。データやアプリケーションやサービスを糊付けて一つの有機体として動作させるのが、スクリプティング言語なのである。

2. Compact Topic Maps

CTM(Compact Topic Maps)の仕様について厳密には考えず、考え違いも恐れず、想像力の翼を拡げて、更新日記をCTMで記述する方法を考えてみよう。トピック(topic)、連想・連合(association)と事象(occurrences)という主要な三要素で日記はどのように表現できるか。

更新日記を構成する要素を列挙してみよう。

・カテゴリ(カテゴリ年間記事リストへのリンク)
・タイトル(記事のアンカー、アンカーのURLはカテゴリキーと記事生成時の時刻を含む)
・パラグラフ(最初のパラグラフRSS1.0のdescriptionに用いられる。Webや更新日記の他の記事へのリンクなどを含む)
・リスト(リンクを含む場合がある)
・表(リンクを含む場合がある)
・写真、図などの画像(画像データへのURLとタイトル、説明のセンテンスを含む)
・更新日時

日記記事のtopicとして取り上げられるのは、まずは記事のタイトルだろう。6月2日の記事を一つ取り上げて考えよう。

6/2/2012 (Sat.)

[マイクロソフト] Windows 8

スティーブン・シノフスキー氏インタビュー：開発責任者に聞くWindows 8の世界――「2年後、タッチできないPCは欠陥品に思われる」 (1/3) - ITmedia +D PC USERネタ。大体の疑問に答えてくれている・・・これはよい記事だね。

Windows 8とWindows RT(ARM版)にはWindowsとして基本的に差はないということがはっきりした。デスクトップも2年後にはタッチして使うことになると想定している。

これらのことが意味していることは、PCだけでなく、ディスプレイにも新たな費用が発生することを意味している。また、以前既に懸念したように、 Windows RTデバイスはiOSタブレット、iPadより

も高価になることは間違いないし、消費電力も大きくなるだろう。コストか、パフォーマンスかという選択に持ち込めるかどうかが勝敗の分かれ目かな。

- Microsoft Windows Developer Days：Windows Store と Windows 8 の潜在的可能性を日本で紹介（1/2） - ITmedia +D PC USER
- 本田雅一のクロスオーバーデジタル：解像度の呪縛から逃れようとする Windows 8（1/2） - ITmedia +D PC USER

アップルの Mountain Lion の方向性が影響を受けるのかどうか、WWDC2012 の成り行きが大変楽しみだ。

更新: 2012-06-02T11:30:43+09:00

HTML は次のような記述になっている。

```
<DT>6/2/2012 (Sat.)<DD>
<div class="emph"><A HREF="renewal_index.html#ms">[マイクロソフト]</A> <A
NAME="ms_1338599791">Windows 8</A></div>
<p><a href="http://plusd.itmedia.co.jp/pcuser/articles/1204/26/news090.html">スティーブン・シノフスキー氏インタビュー：開発責任者に聞く Windows 8 の世界――「2 年後、タッチできない PC は欠陥品に思われる」（1/3） - ITmedia +D PC USER</a>ネタ。大体の疑問に答えてくれている・・・これはよい記事だね。</p>
<p>Windows 8 と Windows RT(ARM 版)には Windows として基本的に差はないということがはっきりした。デスクトップも 2 年後にはタッチして使うことになると想定している。</p>
<p>これらのことが意味していることは、PC だけでなく、ディスプレイにも新たな費用が発生することを意味している。また、以前既に懸念したように、Windows RT デバイスは iOS タブレット、iPad よりも高価になることは間違いないし、消費電力も大きくなるだろう。コストか、パフォーマンスかという選択に持ち込めるかどうかが勝敗の分かれ目かな。</p>
<ul>
<li><a href="http://plusd.itmedia.co.jp/pcuser/articles/1204/25/news025.html">Microsoft Windows Developer Days：Windows Store と Windows 8 の潜在的可能性を日本で紹介（1/2） - ITmedia +D PC USER</a>
<li><a href="http://plusd.itmedia.co.jp/pcuser/articles/1204/12/news074.html">本田雅一のクロスオーバーデジタル：解像度の呪縛から逃れようとする Windows 8（1/2） - ITmedia +D PC USER</a>
</ul>
<p>アップルの Mountain Lion の方向性が影響を受けるのかどうか、WWDC2012 の成り行きが大変楽しみだ。</p>
<div class="updated">更新: 2012-06-02T11:30:43+09:00</div>
```

CTM で書くと、例えば次のようになる。

```
def reference-link($title, $uri)
    title: $title;
    uri: $uri.
end
```

http://homepgae1.nifty.com/kazuf/renewal.html#ms_1338599791
title: "Windows 8";
category: "マイクロソフト";
date: 2012-06-02;
description: reference-link(title: "スティーブン・シノフスキー氏インタビュー：開発責任者に聞く Windows 8 の世界――「2 年後、タッチできない PC は欠陥品に思われる」 (1/3) - ITmedia +D PC USER, uri: http://plusd.itmedia.co.jp/pcuser/articles/1204/26/news090.html);"ネタ。大体の疑問に答えてくれている・・・これはよい記事だね。";
paragraph: "Windows 8 と Windows RT(ARM 版)には Windows として基本的に差はないということがはっきりした。デスクトップも 2 年後にはタッチして使うことになると想定している。";"これらのことが意味していることは、PC だけでなく、ディスプレイにも新たな費用が発生することを意味している。また、以前既に懸念したように、Windows RT デバイスは iOS タブレット、iPad よりも高価になることは間違いないし、消費電力も大きくなるだろう。コストか、パフォーマンスかという選択に持ち込めるかどうかが勝敗の分かれ目かな。";"アップルの Mountain Lion の方向性が影響を受けるのかどうか、WWDC2012 の成り行きが大変楽しみだ。";
list: reference-link(title: "Microsoft Windows Developer Days：Windows Store と Windows 8 の潜在的可能性を日本で紹介 (1/2) - ITmedia +D PC USER", uri: http://plusd.itmedia.co.jp/pcuser/articles/1204/25/news025.html);reference-link(title: "本田雅一のクロスオーバーデジタル：解像度の呪縛から逃れようとする Windows 8 (1/2) - ITmedia +D PC USER", uri: http://plusd.itmedia.co.jp/pcuser/articles/1204/12/news074.html);
updated: 2012-06-02T11:30:43+09:00.

日記記事を、以上のような記述に変換することにどれほどの実用的な意味があるかはまだ疑問な点がある。基本的には、HTML の記述から機械的に容易に読み取れる程度のものでしかないからだ。そのような次第で、なかなか足を前に踏み出せないのが実情である。想像力の翼が閉じたままだ。

本稿を書くにあたり、Compact Topic Maps の仕様について参考にしたのは、次のリンクにある文書である。

ISO/IEC 13250-6: 2010, Information technology - Topic Maps - Part 6: Compact Syntax
http://www.isotopicmaps.org/ctm/ctm.html

個々の記事の記述形式よりは、もっと意味的な解析を全記事について行い、記事の連環を表示するような方向性のほうがおもしろいかもしれない。

日記記事間の意味的なつながりは、通常直接的には日記記事には記載はない。引用でもしない限りは。日記を検索して関連記事のリンクのリストを日記に記載することが、最近では多くなっているが、その作業を組織的・機械的に行うような仕組みを作っておくと役に立つと考えられる。

(投稿: 2012 年 6 月 9 日)

編集後記

jscripter

ようやく編集後記に辿り着いた。

編集から話は変わるが、今日は、結果としてCATVの二台のチューナーをHDMI接続の新しいものに取り替えることになり、CATV経由のメタルプラス電話の契約を追加することになった。

本来、CATV契約を二台を一台にしてコストダウンし、さらにiLink対応のチューナーに切り替えてnasne導入に備えようとしていたのだが・・・実は関係者^^;)が地上波デジタルだけでなくBSも見ていたことが工事後にわかって、連絡したところ、再契約で手数料が必要になることがわかって、がっくり。チューナーがないと地上波デジタルしか見れないよと念押ししていたのに、CATV独自のチャネルが見れないだけとの思い込み。仕方がない。iLinkにはnasneが対応していないこともわかって二重にがっくり。すべて無料だったはずなのに、契約を元に戻して再契約料が必要とは・・・とほほ

CATVの営業から契約解除のキャンセルという取り扱いにしましょうとの電話があって、工事の人から紹介のあったメタルプラス電話が話題になった。インターネット経由の電話は、家内から相手にこちらの声が聞こえないというクレームが付いて困っていたので、切り替えることに。NTTよりは安いし、テレビ契約とセットでほんのわずかに割引もある。電話の置き場所もリビングに戻せる。

iLinkはソニーは積極的に取り扱っていないらしい。チューナーはパナソニック。このようなメーカー間の互換性問題はユーザーの利益にならない。自由に機器類を組み合わせて使うことができないのは大いに困るのである。ビジネス上はコストや販売戦略上の問題があるのだろうけど、もっと他のところで大いに競って欲しいものだ。

それはともかく、さて、暑い夏が来る。皆様、お体にはお気をつけて。

（投稿：2012年6月9日）

ＴＳＮＥＴスクリプト通信

ISSN 1884-2798　出版地: 広島市

2012 年 6 月　9 日　　4.4.001 版
2012 年 6 月 14 日刊行　　4.4.002 版

投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと

編集委員会(投稿順)

機械伯爵　kikwai[at]livedoor[dot]com
ムムリク　qublilabo[at]gmail[dot]com
jscripter　jscripter9[at]gmail[dot]com

著作権

使用許諾・配布条件

免責事項

編集ソフトウェア

OpenOffice.org 3.2.1 Writer

発行所

一次配布所: TSNET スクリプト通信刊行リスト
http://text.world.coocan.jp/TSNET/?TSNET%E3%82%B9%E3%82%AF%E3%83%AA%E3%83%97%E3%83%88%E9%80%9A%E4%BF%A1%E5%88%8A%E8%A1%8C%E3%83%AA%E3%82%B9%E3%83%88

TSNETスクリプト通信 第4巻第4号(通算第16号)
発行: TSC編集委員会 発行日: 2012年 6月 10日
ISSN: 1884-2798 出版地: 広島市 創刊: 2008年5月7日