TSNET スクリプト通信

TSC 編集委員会

ISSN 1884-2798

目次

表紙写真: 奇妙な色彩の夜
撮影: jscripter
日時: 2012 年 7 月 28 日
場所: 広島県立大学付近の路上にて
メモ: なぜ、このような超現実的な色彩が得られたのか、花火大会に向かう道すがら、月のあかりを捉えようとして。

巻頭言

jscripter

はて、混迷を深める世界のなかで、TSNET スクリプト通信の季刊 5 サイクル目に入る第 17 号の意義はどこにあるのだろう。5 サイクル目のはじまりが 5 カ月遅れとなっている。息切れ気味であることは間違いないか。今号のメンバーは、前号と同じだと後付けを編集しながら思った。

今号は、機械伯爵氏の「Python の文法」は第 8 回、分量としてはほぼ八合目に到達したところだそうだ。『コレクションは、Python3 初出の「集合」で終わりで、残りは「組み込み関数」「その他のオブジェクト」で一区切り（本１冊分？）になります』とのこと。

ムムリクさんの「 REXML を使って XML ファイルから本文テキストを抜き出す」は興味深い。XML の解析のニーズは大きいからだ。実は今回、RSS/Atom を Perl のライブラリの XML::Atom や XML::FeedPP で処理して、iUI のデータを出力することを考えたのだが、解析自体がうまくいかない。データ自体が悪いのかなと思うのだが、ブラウザなどでは適切に処理して表示できる。なかなかむずかしいものだ。

僕自身は、「Zed と Lua」と「iUI 入門 - 更新日記モバイル版」の二編を投稿した。「Zed と Lua」は、Zed に組み込まれた Lua を使って何ができるのかを調べたいと思ったのが動機で、初期の報告として書いた。まだはじめたばかりだけど、Lua についてももう少し深く勉強したいと考えている。「iUI 入門 - 更新日記モバイル版」は、JavaScript を本格的に勉強するきっかけにしたいと取り組んでいる。更新日記のモバイル版を作るという実利を兼ねることによって、モチベーションを維持している。

最近は、Google の Go とか dart、マイクロソフトの TypeScript、Joe Hewitt の Up など、新しい言語が出てきている。

・Go（プログラミング言語） - Wikipedia:http://ja.wikipedia.org/wiki/Go_(%E3%83%97%E3%83%AD%E3%82%B0%E3%83%A9%E3%83%9F%E3%83%B3%E3%82%B0%E8%A8%80%E8%AA%9E)
・Dart: Structured web apps:http://www.dartlang.org/
・Welcome to TypeScript:http://www.typescriptlang.org/
・joehewitt/up - GitHub:https://github.com/joehewitt/up

Dart や TypeScript は JavaScript にコンパイルして使うのが目的のようだ。Go は C++の置き換えを狙っているようで、ロブ・パイク、ケン・トンプソン らが開発している。Up はまだ調べていないが、iUI や Firebug を開発した Joe Hewitt のプロジェクトだけに興味がある。

まだまだ、プログラミングの世界は活発に動いている。どこに行き着くのか、見届けたいし、TSNET スクリプト通信を通じて関わっていきたいと考えている。

（投稿日 2012 年 10 月 7 日）

Python の文法

TSNET スクリプト通信版　草稿　第８回

機械伯爵

6-2-2-2-4 写像型(mapping types)

写像型(mapping types)は順列型と異なり、キーとなる文字列などの不変オブジェクトによってアイテムにアクセスすることができるコレクションです。

```
>>> x = {"a":0, "b":1, "c":2}
>>> x["a"]
0
>>> x["b"]
1
>>> x["c"]
2
>>> x["c"]=200
>>> x
{'a': 0, 'c': 200, 'b': 1}
>>> del x["c"]
>>> x
{'a': 0, 'b': 1}
```

写像型は順序を記憶しません（別の言語ではハッシュとも呼ばれます）ので、収納順序は適当に入れ替わります。

現在、写像型としてユーザが使用する目的で組み込みで実装されているのは辞書型だけですが、以下のコードを見てください。

```
>>> class K: A, B = 100, 200
...
>>> K.A                             # "A"はクラス"K"の属性
100
>>> K.__dict__
dict_proxy({'A': 100, '__module__': '__main__', 'B': 200, '__dict__': <attribute
'__dict__' of 'K' objects>, '__weakref__': <attribute '__weakref__' of 'K' objects>,
'__doc__': None})
>>> K.__dict__["A"]                 # "A"というキーで参照できる
100
>>> type(K.__dict__)                # オブジェクトタイプを確認
<class 'dict_proxy'>
>>> K.__dict__["A"] = 1000          # 'dict_proxy'型は読み取り専用
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'dict_proxy' object does not support item assignment
>>> K.A = 1000                      # 属性としてなら変更可能
```

クラス属性は、'dict_proxy'という写像型オブジェクトで参照可能であることがわかります。

'dict_proxy'は参照専用（書き込みはできない）ですが、「辞書的に」属性にアクセスすることが可能です。

辞書は、参照・更新可能なユーザ用の写像型クラスです。

基本的には、キーと値のペア（これをアイテムと言います）でセパレータのコロン':'をはさみ、カ

ンマ','で区切って書きます。

```
>>> d = {"a":100, "b":200, "c":300}
>>> d
{'a': 100, 'c': 300, 'b': 200}
```

キーは、文字列のみならず、不変オブジェクトであれば何でも使用できます。
例えば、タプルは不変オブジェクトですので、次のような妙なことも可能です（この書き方は、リテラルの『内包表記』を参照のこと）

```
>>> mt = {k: i for k, i in zip(
...   ((x, y) for x in range(3) for y in range(3)),
...   range(9)
...   )
... }
>>> for x in range(3):
...   print([mt[x, y] for y in range(3)])
...
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]
```

これは、二次元配列（行列）のようなモノを辞書で表現して、その一行を表示したり、リストに切り取ったりしています。

キーワード引数は、関数の仮変数に'**'をつけて書くことで、辞書として受け取れます。
辞書クラス自身である'dict'は、キーワード引数からダイレクトに辞書を生成します。

```
>>> dict(a=100, b=200, c=300)
{'a': 100, 'c': 300, 'b': 200}
```

キーを文字列で書くのは、クォートやダブルクォートを打つのが面倒なのですが、上記の方法であれば早くすっきり書けます。
さて次は、インスタンス属性が__dict__属性の中に辞書として収納されていることを確認する、ちょっとした実験です。

```
>>> o = type('',(),{})()                  # 空のクラスのオブジェクトを作る裏技です
>>> d = dict(a=100, b=200, c=300)         # 辞書を作ります
>>> o.__dict__ = d                        # 属性を一気に登録します
>>> o.a                                   # 属性参照できます
100
>>> o.x = 400                             # 属性を追加すると・・・
>>> d                                     # 辞書のアイテムが追加されています
{'a': 100, 'x': 400, 'c': 300, 'b': 200}
```

もう一つ、名前空間自体が辞書で実装されていることを確認する実験です。
'vars'関数は、引数無しで呼び出されると、その名前空間に定義されている名前とオブジェクトを返します。

```
>>> nsd = vars()
>>> x               # "x"という変数は定義されていません
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> nsd["x"]=100    # "x"を名前空間の辞書に定義
>>> x               # 変数"x"が登録されています
100
>>>
```

Python では、名前→実体の対応のリストを、辞書型や、辞書型のような型のオブジェクトで管理しています。
このように写像型／辞書オブジェクトは、Python のシステム部分でも利用されています。

※注意：上記の例は、Python のシステムを確認するための実験で、特に'vars'関数を使った凝った裏技は、将来的に使用できなくなる可能性もありますので、使い捨てのスクリプト以外での使用は避けたほうがいいでしょう

6-2-2-2-4-1 辞書(dict)

■クラス名：'dict'

辞書(dict=dictionary)は、他言語では連想配列、ハッシュなどと呼ばれるオブジェクトです
(JavaScript では、辞書を「オブジェクト」と呼んでいるようです)
仕組みや書き方に関しては、前項「写像型」にも少し書いてありますので、参照してください。

※辞書での「アイテム」とは「キー」と「値」のペアを指します。

●通常のメソッド

■アイテム全消去

書式：D.clear() ⇒ None
動作：辞書"D"のアイテムを全消去する

■辞書のコピー

書式：D.copy() ⇒ newD
動作：辞書"D"の内容をコピーした新しい辞書"newD"を返す

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> dd = d.copy()
>>> dd       # コピーされた辞書
{'a': 100, 'c': 300, 'b': 200}
>>> dd.clear()
>>> dd       # 内容をクリア
{}
>>> d        # コピー元には影響なし
{'a': 100, 'c': 300, 'b': 200}
```

■辞書の更新

【辞書による追加／更新】

書式：D.update(Dic)

動作：辞書"Dic"の内容が、辞書"D"に追加（あるいは更新）される（"Dic"は、正確にはキーを持つオブジェクト）

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> d.update({"a":111, "x":666, "y":777})
>>> d
{'a': 111, 'c': 300, 'b': 200, 'y': 777, 'x': 666}
```

【ペアのシーケンスによる追加／更新】

書式：D.update(PSeq)

動作：辞書"D"に、"PSeq"の内容が追加／更新される。"PSeq"は、[[key1, value1], (key2, value2)]などの、ペアのシーケンス（シーケンスはリストでもタプルでも、あるいは文字列等でも可）

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> d.update([["a",111], ("x", 666), "yz"])
>>> d        # "yz"は"y"と"z"のペアとみなされる
{'a': 111, 'x': 666, 'c': 300, 'b': 200, 'y': 'z'}
```

【キーワード引数の開包】

書式：D.update(Key=Val[,Key2=Val2...])

動作：辞書"D"を、キーワード引数により追加／更新

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> d.update(a=111, x=666, y=777)
>>> d
{'a': 111, 'c': 300, 'b': 200, 'y': 777, 'x': 666}
```

■アイテムを取り出し（キー指定）

書式：D.pop(Key) ⇒ Value

動作："Key"で指定したキーの値"Value"を返す。指定されたアイテムは消去される

※注意：'list.pop()'と異なり、キー指定が必須です

■アイテム取り出し（ランダム）

書式：D.popitem() ⇒ Item

動作：アイテムをランダムに選び、キーと値のタプル"Item"を返す

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> d.pop("c")
300
>>> d
{'a': 100, 'b': 200}
>>> d.popitem()
('a', 100)
>>> d
{'b': 200}
```

●特殊な値取得

辞書は通常、その辞書が持っていないキーでアクセスされた場合'KeyError'を返します。しかし、次の二つのメソッドでアクセスされた場合は、エラーを返さず、指定された値を返します
('setdefault'メソッドでは、アイテムを追加してしまいます)

■存在しないキーでもエラーを返さない値取得

書式：D.get(Key[, Default]) ⇒ Value or None[or Default]
動作：辞書"D"のキーに"Key"があれば値"Value"を返す。無ければ'None'か、あるいは"Default"が設定されいれば"Default"を返す

■存在しないキーでアクセスされると、アイテムを追加する値取得

書式：D.setdefault(Key[, Default]) ⇒ Value or None[or Default]
動作：辞書"D"のキーに"Key"があれば値"Value"を返す。無ければ'None'か、あるいは"Default"が設定されいれば"Default"を返す。さらに返したオブジェクトを値として、参照したキーのアイテムが追加される

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> d.get("a")
100
>>> d.get("x")      # エラーは返らない
>>> d.get("x", 999) # デフォルト値を指定するとそれを返す
999
>>> d.setdefault("x")
>>> d               # アイテムが追加されている
{'a': 100, 'x': None, 'c': 300, 'b': 200}
>>> d.setdefault("y",999)
999
>>> d               # アイテムが追加されている
{'a': 100, 'x': None, 'c': 300, 'b': 200, 'y': 999}
```

※通常、辞書に無い値を求められて処理を止めないためには、エラーをフックするか、あらかじめキーがあるかどうかのメンバーテストを行うのが普通ですが、'get'メソッドや'setdefault'メソッドでは、その手間が省けます。なお、'collection'モジュールの'defaultdict'は、存在しないキーを参照した際の対処方法を指定することのできる辞書です。標準モジュールの説明は本書の範囲を超えているので詳しくは説明しませんが、以下のコードを参考にしてください

```
>>> from collections import defaultdict
>>> dd = defaultdict(lambda:999, a=100, b=200, c=300)
>>> for x in dd:
...   print((x,dd[x]))
...
('a', 100)
('c', 300)
('b', 200)
>>> dd["x"]
999
>>> for x in dd:
...   print((x,dd[x]))
...
('a', 100)
('x', 999)
('c', 300)
('b', 200)
```

●演算子や関数呼び出しをフックするメソッド

■アイテム削除(delete an item)

書式：D.__delitem__(Key) ⇒ None
動作：辞書アイテム削除をフックする。上記書式は"del D[Key]"と同じ意味。"Key"はキーとなるオブジェクトを指定する（適合するキーがなければエラーになる）

■インデックス取得

書式：D.__getitem__(Key) ⇒ Value
動作："D[Key]"で呼び出され、キー"Key"の値"Value"を返す

■メンバーテスト

書式：D.__contains__(Key)
動作："Key in D"で呼び出され、辞書の「キー」のメンバーテスト行う。"D"のキーに"key"があれば'True'を、無ければ'False'を戻す

```
>>> d = {"a":100, "b":200, "c":300}
>>> d.__contains__("a")
True
>>> d.__contains__(100)
False
>>>
>>> "a" in d # キーを検索
True
>>> 100 in d # 値(value)ではないので注意！
False
```

※値に対するメンバーテストは、後に説明する'values'メソッドを応用して行います。

■反復子取得

書式：D.__iter__() ⇒ itr
動作："iter(s)"で呼び出され、辞書"D"の反復子"itr"を戻す。また、"for"キーワードによっても自動的に呼ばれて、全てのキーを検索する

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> di = iter(d)
>>> dii = d.__iter__()
>>> "a" in d
True
>>> "a" in di
True
>>> "a" in dii
True
>>> "a" in d
True
>>> "a" in di       # 反復子は使い切り
False
>>> "a" in dii
False
```

■アイテム数取得

書式：D.__len__() ⇒ l
動作："len(D)"で呼び出され、アイテム数を整数型"l"を戻す

■キー指定アイテムの追加、更新

書式：D.__setitem__(Key, Value)

動作："D[Key] = Value"で呼び出され、"Key"が初出なら追加、既出なら更新する

●ビューの取得

読み取り専用の'dict_'と名前のつく「ビュー」を返します。ビューは辞書のキーや値やアイテムを保持しているかのように見える反復可能オブジェクト(iteratable object)です。ビューは、メンバーテストや'for-in'構文による全件検査などに使用できますが、変更は不可能です。ただし、オリジナルの辞書を更新した場合は、ビューも変化します。なおキーとアイテムのビューは、集合または
ビューと集合演算を行うことが可能です

■アイテムリスト取得

書式：D.items() ⇒ DI

動作：辞書"D"のアイテム（のタプル）を参照できる'dict_items'型オブジェクトの"DI"を返す

■キーリスト取得

書式：D.keys() ⇒ DK

動作：辞書"D"のキーを参照できる'dict_keys'型オブジェクトの"DK"を返す

■値リスト取得

書式：D.values() ⇒ DV

動作：辞書"D"の値を参照できる'dict_values'型オブジェクトの"DV"を返す

```
>>> d
{'a': 100, 'c': 300, 'b': 200}
>>> dk = d.keys()   # キーのビュー
>>> dv = d.values() # 値のビュー
>>> di = d.items()  # アイテムのビュー
>>> for x in dk,dv,di:
...   print(x)
...
dict_keys(['a', 'c', 'b'])
dict_values([100, 300, 200])
dict_items([('a', 100), ('c', 300), ('b', 200)])
>>> 100 in dv              # 値のメンバーテストはvaluesで行う
True
>>> 101 in dv
False
>>> d.pop('a')             # キー"a"のアイテムを削除すると・・・
100
>>> d
{'c': 300, 'b': 200}
>>> for x in dk,dv,di:     # ビューからもアイテムが削除されている
...   print(x)
...
dict_keys(['c', 'b'])
dict_values([300, 200])
dict_items([('c', 300), ('b', 200)])
>>> d0 = dict(a=100, b=200, c=300)
>>> d1 = dict(a=100, b=300, x=500)
>>> d0.keys() & d1.keys() # キーの積集合
{'a', 'b'}
>>> d0.items() & d1.items()      # アイテムの積集合
{('a', 100)}
>>> d0.keys() | d1.keys() # キーの和集合
{'a', 'x', 'c', 'b'}
>>> d0.keys() ^ d1.keys() # キーの対称差
{'x', 'c'}
>>> d0.keys() - d1.keys() # キーの差集合
{'c'}
```

●静的メソッド

■順序列型より辞書を新たに生成(クラスメソッド)

書式：dict.fromkeys(S[,v]) = D

動作：順序列型の"S"をキーとした辞書"D"を作成。デフォルト値"v"が設定されている場合は値は全て"v"の値に、設定されていなければ、値は全て'None'になる

```
>>> d = dict.fromkeys("abc")
>>> d
{'a': None, 'c': None, 'b': None}
>>> d = dict.fromkeys("abc",100)
>>> d
{'a': 100, 'c': 100, 'b': 100}
```

(投稿日 2012 年 9 月 17 日)

REXML を使って XML ファイルから本文テキストを抜き出す

ムムリク

視覚障害者のためのデジタル図書の規格である、デイジー図書というものがあります。その詳細については触れませんが基本的に XML ファイルで構成され、音声ファイルをセットにしたり、あるいは TTS 機能を使ってテキストから合成音声を逐次生成して読み上げるなどの機能を持った図書です。

この図書データから本文テキストだけを抜き出す必要ができたので、REXML を使って試してみました。

当該図書ファイルでは、拡張子は html となっていますが、デイジー図書では読み上げにあたって現在読んでいる文章部分をハイライト表示するため、テキストはフレーズごとに span タグで囲まれています。また、ウェブ上に公開されるページという用途ではないため、各種のスクリプトなどが組み込まれることもないので REXML でも特に問題なく処理できるようです。（URL に「&」が入っていると、REXML では不正な文字の使用としてエラーになるようです）

ここでは標準ライブラリとして収められているため手軽であるということから REXML を使用していますが、もしもウェブページなどの HTML を処理したいというのであれば Nokogiri などを使用するほうがより適切かもしれません。

■まずは試してみる

まずはシンプルなものでどのような結果が得られるのか試してみます。サンプル図書として無料で提供されているマルチメディアデイジー図書「ごん狐」を利用します。
（http://www.normanet.ne.jp/services/download/daisy.html）

```
----- sample1.rb -----
1: require 'rexml/document'
2: def get_elements(elm)
3:     elm.each_element {|e|
4:         p e
5:     }
6: end
7: doc = REXML::Document.new(File.open("gon01.html"))
8: body = doc.elements["/html/body"]
9: get_elements(body)
----------------------
```

図書ファイルのあるフォルダでスクリプトを実行すると次のような結果が表示されます。

```
<div class='tate'> ... </>
```

（結果：1）

sample1 では gon01.html を開いて(7)、body 要素を起点として(8)、それ以下の要素を順に表示させます(3,4)。

しかし、表示されたのは「 ... </>」だけでした。つまり、body の直下の大きなくくりには しかなく、その中にその他のものがはいっているということのようです。

そこで、sample1.rb に少し追加して繰り返し要素をたどるようにしてみます。

```
----- sample1.rb -----
1: require 'rexml/document'
2: def get_elements(elm)
3:    elm.each_element {|e|
4:       p e
a:       if e.has_elements?
b:          get_elements(e)
c:       end
5:    }
6: end
7: doc = REXML::Document.new(File.open("gon01.html"))
8: body = doc.elements["/html/body"]
9: get_elements(body)
----------------------
```

4 と 5 の間に a、b、c、を追加しました。もしもまだ要素を中に持っているのであれば(a)、取得する(b)、というものです。実行してみます。

```
<div class='tate'> ... </>
<div id='honbun'> ... </>
<h2 id='njpu_0004'> ... </>
<a href='njpu0002.smil#njpu_0004'> ... </>
<p class='indent_top'> ... </>
<span id='xnjp_0002'> ... </>
<ruby> ... </>
<rb> ... </>
<rp> ... </>
<rt> ... </>
<rp> ... </>
<ruby> ... </>
<rb> ... </>
<rp> ... </>
<rt> ... </>
<rp> ... </>
<br/>
<span id='xnjp_0003'> ... </>
 (以下続く)
        (結果：2)
```

実際の gon01.html で確認してみます。

```
<div class="tate"><div id="honbun">
```

```
        <h2 id="njpu_0004"><a href="njpu0002.smil#njpu_0004">一</a></h2>

        <p class="indent_top"> <span id="xnjp_0002">これは、<ruby><rb>私</rb><rp> (</rp><rt>
わたし</rt><rp>) </rp></ruby>が小さいときに、村の<ruby><rb>茂平</rb><rp> (</rp><rt>もへい
</rt><rp>) </rp></ruby>というおじいさんからきいたお話です。</span><br />
        <span id="xnjp_0003">  むかしは、私たちの村のちかくの、<ruby><rb>中山
</rb><rp> (</rp><rt>なかやま</rt><rp>) </rp></ruby>というところに小さなお城があって、中山さ
まというおとのさまが、おられたそうです。</span>
        </p>
        （以下続く）
```

タグの並びは間違いないようです。これですべての要素をたどることができそうです。

■テキストを取得してみる

要素の中のテキストは element.text で取得できます。sample1.rb の 4 の「p e」を「puts e.text」と替えて実行してみます。

```
一

これは、

私
（
わたし
）

茂平
（
もへい
）

  むかしは、私たちの村のちかくの、

中山
（
なかやま
）



  その中山から、少しはなれた山の中に、「ごん
（以下続く）
        （結果：3）
```

なんだかおかしなことになりました。

はじめの「これは、 私 （ わたし ） 茂平 （ もへい ）」というのは、実際には、次のようなものです。

これは、<ruby><rb>私</rb><rp>（</rp><rt>わたし</rt><rp>）</rp></ruby>が小さいときに、村の<ruby><rb>茂平</rb><rp>（</rp><rt>もへい</rt><rp>）</rp></ruby>というおじいさんからきいたお話です。

結果：2 を見直すとわかってきますが、span の中に ruby があるために、このままでは全体を取得することがうまくできないようです。このような場合に、その要素のテキストすべてを取得する element.texts があります。試してみます。

さきほどの「puts e.text」を「puts e.texts」に修正して実行します。

```
ー



これは、
が小さいときに、村の
というおじいさんからきいたお話です。
私
（
わたし
）
茂平
（
もへい
）
　むかしは、私たちの村のちかくの、
というところに小さなお城があって、中山さまというおとのさまが、おられたそうです。
（以下続く）
        （結果：4）
```

「これは、」から「もへい　）」までを見ると、どうやら必要なテキストそのものはすべてあるようですが、順序は正しくありません。

つまり、element.texts で取得されるテキストは、その要素直下で間を他の要素などで区切られてしまっているものを、配列として取得するということのようです。

テキスト内にはいる要素として考えられるのは、ルビ（ruby）や文字修飾（strong, em など）、リンク（a）あたりでしょうか。とりあえずここで取り上げているような物語ではせいぜいその程度で済みそうです。

■テキストを正しい順序に直す

では、どうやってその順序を正しくするのか。children メソッドが使えそうです。先の部分の children メソッドによる結果を見ると、次のようになっています。

```
        ["¥u3053¥u308C¥u306F¥u3001", <ruby> ... </>,
"¥u304C¥u5C0F¥u3055¥u3044¥u3068¥u304D¥u306B¥u3001¥u6751¥u306E", <ruby> ... </>,
```

"¥u3068¥u3044¥u3046¥u304A¥u3058¥u3044¥u3055¥u3093¥u304B¥u3089¥u304D¥u3044¥u305F¥u304A¥u8A71¥u3067¥u3059¥u3002"]

得られた文字が UTF-8 ですが、

["これは、",<ruby>...</>,"が小さいときに、村の",<ruby>...</>,"というおじいさんからきいたお話です。"]

となっているらしきことがわかります。このとき、それぞれの文字は REXML::Text クラスのオブジェクトなので、それを頼りに正しい順序に直してみることにします。

```
----- restore_doc -----
A: def restore_doc(elm)
B:     docs = elm.texts
C:     str = ""
D:     i = j = 0
E:     elm.children.each {|c|
F:         if c.instance_of?(REXML::Text)
G:             str << docs[i].to_s
H:             i += 1
I:         else
J:             str << @modify[j].to_s
K:             j += 1
L:         end
M:     }
N:     @modify = []
O:     return str
P: end
-----------------------
```

ルビを含んだ span 要素を A に与えます。span 要素直接のテキストすべてを docs に入れます（B）。

span 要素の children 要素の集合を順に処理していき（E）、もしもそれが REXML::Text オブジェクトであるならば（F）、str に docs の文字を追加します（G）。docs のインデックスを進めておきます（H）。

REXML::Text 以外であれば（I）、あらかじめルビなどを収めておいた @modify という配列から str に文字を追加します（J）。こちらも同様にインデックスを進めます。

最後に @modify 配列を初期化して（N）、できあがった文字列 str を返します（O）。

これを含めたすべてのスクリプトを実行すると結果は次のようになりました。

```
ー
    これは、私《わたし》が小さいときに、村の茂平《もへい》というおじいさんからきいたお話です。
      むかしは、私たちの村のちかくの、中山《なかやま》というところに小さなお城があって、
```

中山さまというおとのさまが、おられたそうです。
その中山から、少しはなれた山の中に、「ごん狐《ぎつね》」という狐がいました。ごんは、一人《ひとり》ぼっちの小狐で、しだの一ぱいしげった森の中に穴をほって住んでいました。そして、夜でも昼でも、あたりの村へ出てきて、いたずらばかりしました。はたけへ入って芋をほりちらしたり、菜種《なたね》がらの、ほしてあるのへ火をつけたり、百姓家《ひゃくしょうや》の裏手につるしてあるとんがらしをむしりとって、いったり、いろんなことをしました。
（以下続く）
（結果：5）

よさそうです。

■ ■ ■ ■

実のところは、再帰処理しているためか、テキストの終わりのほうに大量の改行が含まれてしまったりするのですが、ここではそれはよしとしておきます。そのほか、まだ改善したい点はあるかもしれませんが、比較的手軽にテキストそのものとして取り出すことができました。

データベース的な XML ファイルのように、構造がしっかりと決まっているものであれば、もっと簡単に扱えるでしょう。

標準添付ライブラリとして手軽に使えるという魅力は、もっと活用してよいのかもしれません。

参考サイト：
私家版REXML API リファレンス
(http://www.cozmixng.org/~kou/ruby/rexml/reference)

```
=============================================================================
# encoding: utf-8
require 'rexml/document'

def get_doc(body)
   body.each_element {|e|
      case e.name
      when 'div'
         other_elements?(e)
      when 'p'
         other_elements?(e)
      when /h[1-6]/
         @modify << "¥n"
         other_elements?(e)
      when 'a'
         other_elements?(e)
      when 'span'
         case e.attributes["class"]
         when 'underline'
            @modify << e.text
```

```
         else
            other_elements?(e)
         end
      when 'ruby'
         e.each_child {|r|
            next if "¥n" == r.to_s
            case r.name
            when 'rb'
               @kanji = r.text
            when 'rt'
               @ruby = r.text
            end
         }
         @modify << "#{@kanji}《#{@ruby}》"
      when 'strong', 'b', 'em', 'i'
         @modify << e.text
      when 'img'
         other_elements?(e)
      when 'br'
         @texts += "¥n"
      else
         other_elements?(e)
      end
   }
end

def other_elements?(elm)
   if elm.has_elements?
      get_doc(elm)
      @texts += restore_doc(elm)
   else
      @texts += elm.text unless elm.text.nil?
   end
end

def restore_doc(elm)
   docs = elm.texts
   str = ""
   i = j = 0
   elm.children.each {|c|
      if c.instance_of?(REXML::Text)
         str << docs[i].to_s
         i += 1
      else
         str << @modify[j].to_s
         j += 1
      end
   }
   @modify = []
```

```
   return str
end


doc = REXML::Document.new(File.open("gon01.html"))
body = doc.elements["/html/body"]
@texts = ""
@modify = []

get_doc(body)
puts @texts
```

==========

（投稿日 2012 年 9 月 28 日）

Zed と Lua

jscripter

はじめに

テキストエディタ Zed は、これまで TSNET スクリプト通信(3,6,8 号)で 3 回に渡って紹介してきたように、標準入出力を持つ Perl などの外部のスクリプティング言語により、編集テキストを加工したり、他のアプリケーションと結びつけることができます。

実は、Zed は二つの言語を内蔵しています。そのうちの一つが Lua です。Lua は作者の一人の Luiz 氏によれば、「組み込み可能で軽量かつ高速でパワフルなスクリプティング言語」(Federico Biancuzzi、Shane Warden 編「言語設計者たちが考えること」、オライリー・ジャパン、2010 年)です。Lua は Zed に組み込まれ、Zed を制御することができます。もちろん、スクリプティング言語自体の機能も使えます。エディタとスクリプティング言語の統合度が上がることによって、情報処理にさらなる新しい展開がもたらせるかどうかを試せるチャンスが訪れています。

作者の久世氏に、lua メモ.txt と Script 仕様.txt の収録を許可いただいたので、付録として本稿の最後に掲載します。lua 言語の仕様については lua メモ.txt を、Zed 独自の命令については Script 仕様.txt を参考にしてください。また、Zed の持つ様々な機能や今後の展開の詳細については Readme.txt を参照してください。

本稿では、内蔵 Lua 言語による編集の方法について簡単に説明します。標準で、小文字を大文字に変換するスクリプト(upper.lua)と大文字を小文字に変換するスクリプト(lower.lua)が、macro¥exec¥menu¥edit のフォルダに添付されているので、編集のメニューから実行できます。そちらも参考にしてください。

選択したテキスト行を HTML のリスト表現に変換して置換出力する例(1)

[hlist.lua]

```
-- Title = HTML List 出力
-- 2012/06/29
require "utils"
E = Editor{}
local Tbl = {}
Tbl = StrToTbl(E.SelText, "[^¥n¥r]+")
E.SelText = "<ul>¥n"
for i = 1, #Tbl do
       E.SelText = "¥t<li>" .. Tbl[i] .. "¥n"
end
E.SelText = "</ul>¥n"
```

1. “--”のようにハイフン(-)を二つ並べると行末までがコメントになります。
2. コメント行の”Title = HTML List 出力”については Script 仕様.txt を参照してください。編集などのメニューのタイトルになります。
3. “require”は外部ライブラリを呼び出します。現在の Zed(Ver.0.64)には、lua のフォルダに

alien.lua、inifile.lua、utils.lua の三つのライブラリが格納されています。後で出てくる'StrToTbl'という文字列をセパレータで分割してテーブルに代入する関数などが定義されています。

4. 'Editor{}'は Editor テーブルを生成するコンストラクタです。テーブルは連想配列の実装であり、オブジェクトです。
5. '{}'はテーブルを生成するコンストラクタです。Tbl というテーブルを生成します。
6. StrToTbl 関数で E.SelText(エディタの選択行)を改行で分割して Tbl テーブルに代入します。
7. 'E.SelText'の'SelText'は Editor テーブル(オブジェクト)のプロパティです。代入で値を設定できます。まず、HTML のリスト開始タグ<ul>を設定します。
8. for ループで Tbl テーブル(連想配列)の値(行の文字列)を 1(最初のキー)から#Tbl(最後のキー)まで HTML リストタグ<li>と改行で挟んで、E.SelText に設定していきます。
9. 最後に、E.SelText に HTML リスト終了タグ
を設定します。

次に別の書き方を示しましょう。外部ライブラリを使わない方法です。

選択したテキスト行を HTML のリスト表現に変換して置換出力する例(2)

[hlist2.lua]

```
-- Title = HTML List 出力 2
-- 2012/07/03
E = Editor{}
local Str = {}
st = E:RowToLines(E.SelSr)
en = E:RowToLines(E.SelEr)
print("Start:", st)
print("End:", en)
for i = st, en do
Str[i] = E:GetLineString(i)
end
E.SelText = "<ul>¥n"
for i = st, en do
	E.SelText = "¥t<li>" .. Str[i] .. "¥n"
end
E.SelText = "</ul>¥n"
```

1. Editor オブジェクトにはメソッドとプロパティがあります。Script 仕様.txt の「Zed 独自の命令表」から該当部分を引用しておきましょう。

・プロパティ

型	名前	Lua	Get	Set	説明
String	SelText	Editor	○	○	選択範囲の文字列
int	SelSr	Editor	○	×	選択範囲の始端位置(行)
int	SelEr	Editor	○	×	選択範囲の終端位置(行)

・メソッド

戻り値	名前	Lua	引数	説明
int	RowToLines	Editor	Row	Row で指定された行の Lines 上の行インデックスを返す
String	GetLineString	Editor	i	指定行の文字列を取得 i-行番号(Line カウントで指定)

2. Lua のオブジェクトのメソッドは、「:」で区切って示し、プロパティは「.」で区切って示します。

3. print 出力は Zed の内蔵コマンドシェルに出力されます。

おわりに

さて、以上で、Lua のスクリプティングと Zed エディタ独自の命令の組み合わせ方の基本的な部分が紹介できたと思います。次に必要なのは、活用のアイデアです。是非、みなさんもチャレンジしてください。Zed は β 版といっても、現時点でも先端的で十分に強力なエディタです。今後、さらに強力なエディタに変貌していくことを期待しています。

付録

[lua メモ.txt]

【Lua のためのメモ】 2012.6.26
少しずつ書き溜めていきます。(Lua 5.2)

■演算子
〇種類
・算術演算子
+ 加算、 - 減算、 * 乗算、 / 除算、 % 剰余
^ 累乗、 - 単項の符号反転
= 代入

・関係演算子
== 等しい、 ~= 等しくない
<= 以下、 >= 以上
< 未満、 > 大

・論理演算子
and 論理積、 or 論理和、 not 否定

・その他
.. 連結：文字列の連結(数値の場合、文字列に変換されて連結される)
長さ演算子：文字列の長さはそのバイト数のことです。
( ) 演算式に利用する。()から演算が行われる。
./[] テーブルのキーの指定に使用。
abc.x =1
abc[x]=1

○優先順位
優先順位は低い方から順に
or
and
< > <= >= ~= ==
..
+ -
* / %
not # - (単項)
^
注: 連結 (`.. ´) と累乗 (`^ ´) は右結合。他の二項演算子はすべて左結合。

■データ型と変数
○変数
変数への型の指定(宣言)は不要。使用したいときに宣言なしで使用可能。
local 変数の指定がある場合は、ローカル変数。ない場合は、グローバル変数となる。
例 local a,b a,b をローカル変数として定義

○データ型の種類
・nil
・論理型
true, false
・数値
整数(int), 浮動小数点数型(double)
16 進数表記: 接頭辞 0x を記述
1
1.2
1.2e3
0x1234 ・・・ 16 進数表記
・文字列
文字列リテラル
'文字列'
"文字列"
[[文字列]]

エスケープシーケンス(文字列リテラル内で使用)
¥a ベル
¥b バックスペース
¥f 改ページ
¥n 改行
¥r 復帰
¥t 水平タブ
¥v 垂直タブ
¥¥ ¥記号
¥" ダブルクォーテーション
¥' シングルクォーテーション
¥ddd ASCII コード指定(ddd は最大 3 桁の 10 進数)。(例 ¥97 は"a"と同じ)
¥xXX ASCII コード指定(XX はぴったり 2 桁の 16 進数)。

・関数

```
function 関数名(パラメータ)
  文
end
```

もしくは、

```
関数名 = function (パラメータ)
  文
end
```

戻り値は、return で指定。(複数指定可能)

可変長引数は、"..."で指定

例 1

```
-- a, b の加算
function Add(a,b)
  return a+b
end
```

例 2

```
-- a, b の加算, 減算
function AddSub(a,b)
  return a+b, a-b
end
```

注： 戻り値は、上記のように複数個指定できます。

例 3

```
-- 全引数の合計
function Sum(...)
  local ret=0
  local i
  local para = {...}
  for i=1,#para do
   ret = ret+para[i]
  end
  return ret
end
```

注： 可変長引数"..."は、テーブルではないため、テーブルにしたい場合は、{}で囲む必要があります。

・ユーザーデータ

Lua 組み込み側のアプリで定義されたデータであり、アプリ側で準備した関数で処理するデータ。

・スレッド

コルーチンにより実行されている処理。(OS のスレッドとは異なる)

・テーブル
{["キー1"=]値1, ["キー2"=]値2, ...}
キー: nil を除く任意の値(省略可。省略時は、自動で1から順に整数値が与えられる。)
値 : nil を除く任意の値
例
{1,2,3,"xyz"} テーブル値が1, 2, 3, "xyz"のテーブル
{1;2;3;"xyz"} テーブル値が1, 2, 3, "xyz"のテーブル
{[1]="a",[3]="b"} キー1の値が"a", キー3の値が"b", キー2はなし
{x="a","b","c"}キーxの値が"a", キー1の値が"b", キー2の値が"c"
{["x"]="a","b","c"} 同上
例2
```
tbl = {x="a","b","c"} -- テーブル初期化
printf(tbl[1].."¥n") -- "b"が表示される
printf(tbl[2].."¥n") -- "c"が表示される
printf(tbl["x"].."¥n") -- "a"が表示される
printf(tbl.x.."¥n") -- "a"が表示される
```

■コメント
○行コメント
```
-- コメント
```
○ブロックコメント
```
--[[
  コメント文
]]
```
■制御構文
○if文
if 「条件1」 then 「プロック1」 {elseif 「条件2」 then 「プロック2」 } [else 「プロック3」] end;
「条件1」が真のとき「プロック1」を実施、偽のとき以降を実施。
「条件2」が真のとき「プロック2」を実施、偽のとき以降を実施。・・・elseifは繰り返し使用可能
すべての条件を満たさなかったとき、「プロック」3を実施。

例
```
if a==0 then
  b=1
  c=1
elseif a==1 then
  b=2
  c=2
else
  b=3
  c=3
end
```

aが、0のときb, cには 1が代入される。
aが、1のときb, cには 2が代入される。
それ以外のとき、b, cには 3が代入される。

○for 文-数値用
for var = e1, e2[, e3] do 「プロック」 end
var=e1 から、e2 まで、e3 刻みで増加しながら、「プロック」を実行する。
e3 省略時は、e3=1 として処理される。

例
```
for i=0,100,2 do
  a=i
end
```

i は、2 刻みで、0 から 100 まで増加し、a に代入されます。

○for 文-汎用
for var_1, ・・・, var_n in explist do block end
イテレータ(explist) と呼ばれる関数を用いる繰り返し命令です。
繰り返しのたびに、 イテレータ関数が呼ばれ var_1～var_n が更新される。
この新しい値が nil になるまで続きます。

例
```
list = {key1=1, key2="b", key3=3}
for key,v in pairs(list) do
  io.write(key.."="..v.."¥n")
end
```

下記のような出力になる。
```
key1=1
key3=3
key2=b
```

○while 文
while 「条件」 do 「プロック」 end
「条件」が成立している間、「プロック」を実行する。

○repeat 文
repeat 「プロック」 until 「条件」
「プロック」を実行後、「条件」が成立している間、再度「プロック」を実行する。

○break 文
for, while, repeat などのループから抜ける

○return 文
関数やチャンクから値を返す。(1 個以上返せる)

注：構文的な理由から、return 文と break 文はブロックの最後の文としてのみ書くことが許される。
return や break をブロックの途中で使うことが本当に必要なら、次のような慣用句を使えば良い。

```
do return end
```

```
    do break end
```

■標準関数

〇基本関数

〇標準/ファイル入出力

・io.write(msg, ...)　　・・・標準出力

・io.read　　・・・標準入力（現在未対応！！）

・io.open([file])　　・・・ファイルをテキストモードで開く

・file:read([para, para, ...])　　・・・書式に従いファイルを読み込む

```
*all   / *a  : 全て読み込む
*line  / *l  : 次の行を読み込む。ファイ終端に達した場合は、nil を返す。（デフォルト）
*number/ *n  : 数値を読み込む。
num          : 指定文字数読み込む。
```

例

データファイル(a.dat)

```
1 2 3
4 5 6
...
```

```
f=io.open("a.dat")
while true do
  local n1, n2, n3 = f:read("*n", "*n","*n")
  if not n1 then break end
  print(n1, n2, n3)
end
f:close()
```

・io.lines([filename])　　・・・１行ずつ読み込みを行うイテレータ
(fileオープン/クローズ自動)

例

```
for line in io.lines("a.dat") do
  print(line)
end
```

・file:lines()　　・・・１行ずつ読み込みを行うイテレータ

例

```
f=io.open("a.dat")
for line in f:lines() do
  print(line)
end
f:close()
```

〇文字列操作

・string.byte(s [, i[, j]])

・string.char(i1, i2, ...)

・string.dump(function)

・string.find(s, pattern [, init [, plain]])

・string.format(書式文字列, e1, e2, ...)

・string.gmatch(s, pattern)
・string.gsub(s, pattern, repl [, n])
・string.len(s)
・string.lower(s)
・string.match(s, pattern [, init])
・string.rep(s, n)
・string.reverse(s)
・string.sub(s, i [, j])
・string.upper(s)
・書式文字列
・pattern
- 特別な文字
. すべての文字
%a すべての大文字、小文字
%c すべての制御文字
%d すべての数字
%g 空白以外のすべての印刷可能文字
%l すべての小文字
%p すべての区切り記号
%s すべての空白文字
%u すべての大文字
%w すべての英数文字
%x すべての16進数字
%. .自身
%% %自身
- 文字集合
[set] set内のすべての文字のいずれか1文字を示す
例: [0-9] 0～9のいずれか1文字を示す
[^set] setの補集合
- 繰り返し
* 0回以上の繰り返し
+ 1回以上の繰り返し
- 0回以上の繰り返し
'*' と異なり、この繰り返し要素は可能な限り 短い シーケンスにマッチする。
? 0回、もしくは、1回
- 後方参照
%n (nは1～9) n番目にキャプチャされた文字列にマッチ
- その他
%b xy (x,yは異なる文字) xで始まってyで終わる文字列
xとyは数の対応が取れている。
つまり、文字列を左から順にサーチし、xが現れるたびにカウントが+1され、
yが現れるたびにカウントが-1され、カウントはじめのxからカウン0までの文字列
例 %b () は、カッコの対応がとれた文字列にマッチする
- キャプチャ
パターンは、カッコ()で囲まれたサブパターンを持つことができ、それらはキャプチャーと呼ばれる。
キャプチャされた部分文字列は、後で%nで使用することができる。
キャプチャの番号は、左から順番に順番付けされる。
特殊なケースとして、空キャプチャ () は文字列の現在位置（数値）をキャプチャする。

例えば、文字列 "abcde" にパターン "()bc()" を適用すると、 2つのキャプチャ 2(数値)と4(数値)が得られる

〇テーブル操作
- table.concat(list [, sep [, i [, j]]])
- table.insert(list, [pos,] value)
- table.pack(・・・)
- table.remove(list [, pos])
- table.sort(list [, comp])
- table.unpack(list [, i [, j]])

〇数学関数

〇ビット演算
- bit32.arshift(x, disp)
- bit32.band(・・・)
- bit32.bnot(x)
- bit32.bor(・・・)
- bit32.btest(・・・)
- bit32.bxor(・・・)
- bit32.extract(n, field [, width])
- bit32.replace(n, v, field [, width])
- bit32.lrotate(x, disp)
- bit32.lshift(x, disp)
- bit32.rrotate(x, disp)
- bit32.rshift(x, disp)

〇コルーチン操作

〇モジュール

〇OS 関連操作

〇デバッグライブラリ

■内蔵ライブラリ

通常の lua5.2 のライブラリと同じ

■外部ライブラリ

〇呼び出し方法

```
require("外部ライブラリ")
```

〇luacom(COM ライブラリ)

例: エクセル操作

```
require("luacom")

local excel = luacom.CreateObject("Excel.Application") -- Excel の起動
excel.Visible = true                                   -- 可視状態に
local wb = excel.Workbooks:Add()                       -- ワークブックを追加
local ws = wb.Worksheets(1)                            -- ワークシートを定義
for row=1,100 do
  ws.Cells(row,1).Value2 = math.random()
  ws.Cells(row,2).Value2 = math.random()
  ws.Cells(row,3).Value2 = math.random()
end
```

○alien(外部ライブラリ(DLL)呼び出しライブラリ)
Lua での関数名 = alien.DLL 名.関数名 で、外部ライブラリを取り出し
Lua での関数名:types{ ret = 戻り値, abi = 呼び出し規約の種類, 第 1 の引数の種類, 第 2 の引数の種類, ... } で、
関数の型を定義する。
これを用いれば、Windows の win32api 群や、サードパーティー製の DLL が利用可能。

例: MessageBox 操作

```
require( "alien" )

local mbox = alien.User32.MessageBoxA
mbox:types{ ret = 'long', abi = 'stdcall', 'long', 'string', 'string', 'long' }
mbox(0, "Hello Lua World!", "Title", 0)
```

○lfs(LuaFileSystem: File System ライブラリ)
例: カレントフォルダ以下のファイルを列挙(属性情報付き)

```
require"lfs"

function attrdir (path)
    for file in lfs.dir(path) do
        if file ~= "." and file ~= ".." then
            local f = path..'/'..file
            print ("¥t "..f)
            local attr = lfs.attributes (f)
            assert (type(attr) == "table")
            if attr.mode == "directory" then
                attrdir (f)
            else
                for name, value in pairs(attr) do
                    print (name, value)
                end
            end
        end
    end
end

attrdir (".")
```

○inifile(ini ファイル操作用ライブラリ) ... alien による win32api の実装

・GetPrivateProfileSection(AppName, ReturnedString, Size, FileName)
AppName : セクション名
ReturnedString : 情報が格納されるバッファ
Size : 情報バッファのサイズ
FileName : .ini ファイルの名前

例

```
require("inifile")
```

```
require("utils")

local ini = '.¥¥ZED.ini'
local bufsize = 128
local buf = alien.buffer(bufsize)

len = GetPrivateProfileSection("環境", buf, bufsize, ini)
nameTbl = StrToTbl(buf:tostring(len))
print(table.concat(nameTbl, ','))
```

・GetPrivateProfileSectionNames(ReturnBuffer, Size, FileName)

ReturnBuffer : 情報が格納されるバッファ
Size : 情報バッファのサイズ
FileName : .ini ファイルの名前

例

```
require("inifile")
require("utils")

local ini = '.¥¥ZED.ini'
local bufsize = 128
local buf = alien.buffer(bufsize)

len = GetPrivateProfileSectionNames(buf, bufsize, ini)
nameTbl = StrToTbl(buf:tostring(len))
print(table.concat(nameTbl, ','))
```

・GetPrivateProfileInt(AppName, KeyName, Default, FileName)

AppName : セクション名
KeyName : キー名
Default : キー名が見つからなかった場合に返すべき値
FileName : .ini ファイルの名前

例

```
require("inifile")

local ini = '.¥¥ZED.ini'

v = GetPrivateProfileInt("環境", "Status", 0, ini)
printf("[環境] Status = %s¥n¥n", v)
```

・GetPrivateProfileString(AppName, KeyName, Default, ReturnedString, Size, FileName)

AppName : セクション名
KeyName : キー名
Default : 既定の文字列
ReturnedString : 情報が格納されるバッファ
Size : 情報バッファのサイズ

FileName : .ini ファイルの名前

例

```
require("inifile")
require("utils")

local ini = '.¥¥ZED.ini'
local bufsize = 128
local buf = alien.buffer(bufsize)

len = GetPrivateProfileString("環境", "History", "NoData", buf, bufsize, ini)
printf("[環境] History = %s¥n¥n", buf:tostring(len))
```

・GetPrivateProfileStruct(Section, Key, Struct, SizeStruct, File)
Section : セクション名
Key : キー名
Struct : 取得した値を受け取るバッファのアドレス
SizeStruct : 取得した値を受け取るバッファのサイズ
File : .ini ファイル

・WritePrivateProfileSection(AppName, String, FileName)
AppName : セクション名
String : 書き込むべきデータ
FileName : ファイル名

・WritePrivateProfileString(AppName, KeyName, String, FileName)
AppName : セクション名
KeyName : キー名
String : 追加するべき文字列
FileName : .ini ファイル

・WritePrivateProfileStruct(Section, Key, Struct, SizeStruct, File)
Section : セクション名
Key : キー名
Struct : 追加するデータが入ったバッファへのポインタ
SizeStruct : バッファのサイズ （バイト数）
File : .ini ファイル

[Script 仕様.txt]

【Script 命令表】 for Zed Ver 0.64

●Script に標準装備されている命令について
Lua : http://milkpot.sakura.ne.jp/lua/lua52_manual_ja.html
などを参考にしてみてください。
Dmonkey : dmsref.hlp
などを参考にしてみてください。

●Zed 独自の命令表の見方

プロパティ：状態を取得/変更できる変数のことです

例: Editor のプロパティー SelText に対し、値 v を取得/設定する

Lua の場合

```
E = Editor{};                          Editor テーブルの作成
v = E.SelText;                         値の取得
E.SelText = v;                         値の設定
```

Dmonky の場合

```
E = new Editor;                        Editor オブジェクトの作成
v = E.abc;                             値の取得
E.abc = v;                             値の設定
```

COM(JScript)の場合

```
var E=new ActiveXObject("zed.App");    オブジェクトの作成
v = E.abc;                             値の取得
E.abc = v;                             値の設定
```

メソッド　:関数です

例: Editor のメソッド abc(x[,y[,z]])

Lua の場合

```
E = Editor{};                          Editor テーブルの作成
ret = E:abc(x)
ret = E:abc(x,y)
ret = E:abc(x,y,z)
#パラメーターが可変です
```

Dmonky の場合

```
E = new Editor;                        Editor オブジェクトの作成
ret = E.abc(x,y,z);
```

COM(JScript)の場合

```
var E=new ActiveXObject("zed.App");    オブジェクトの作成
ret = E.abc(x,y,z);                    値の取得
```

●Zed 独自の命令表

・プロパティ

型	名前	Lua	DM	COM	Get	Set	説明
String	Text	Editor	Editor	○	○	○	全文字列
String	SelText	Editor	Editor	○	○	○	選択範囲の文字列
int	SelStart	Editor	Editor	○	○	○	文字列の先頭から現在のキャレット位置までの文字列長
int	SelLength	Editor	Editor	○	○	○	選択範囲の文字列長
bool	Selected	Editor	Editor	○	○	×	選択状態か
int	SelMode	Editor	Editor	○	○	○	選択モード 0-通常、0 以外-矩形
int	SelSr	Editor	Editor	○	○	×	選択範囲の始端位置(行)
int	SelEr	Editor	Editor	○	○	×	選択範囲の終端位置(行)
int	SelSc	Editor	Editor	○	○	×	選択範囲の始端位置(桁)
int	SelEc	Editor	Editor	○	○	×	選択範囲の終端位置(桁)
int	LineCount	Editor	Editor	○	○	×	全行数
int	RowCount	Editor	Editor	○	○	×	画面上に表示出来る行数

	int	ColCount	Editor	Editor ○	○	×	画面上に表示出来る文字数(桁数)
	int	Row	Editor	Editor ○	○	○	キャレット位置(行)
	int	Col	Editor	Editor ○	○	○	キャレット位置(桁)
	int	TopRow	Editor	Editor ○	○	○	画面上端の行番号
	int	TopCol	Editor	Editor ○	○	○	画面左側の桁番号
	bool	WordWrap	Editor	Editor ○	○	×	折り返し処理を行うかどうか
	int	WrapByte	Editor	Editor ○	○	×	折り返し桁数
	bool	ReadOnly	Editor	Editor ○	○	○	読み取りOnly
	bool	Modified	Editor	Editor ○	○	×	変更されたか
	bool	OverWrite	Editor	Editor ○	○	○	上書きモード
	int	TabSpaceCount	Editor	Editor ○	○	○	タブ数
	String	TokenString	Editor	Editor ○	○	×	カーソル位置のトークン
	String	WordString	Editor	Editor ○	○	×	カーソル位置の単語
	int	Handle	Editor	Editor ○	○	×	エディタ部のハンドル
	String	FindString	Editor	Editor ○	○	○	検索文字列
	String	ReplaceString	Editor	Editor ○	○	○	置換文字列
	bool	FindOptTopOfFile	Editor	Editor ○	○	○	検索条件 ファイルの先頭から
	bool	FindOptMatchCase	Editor	Editor ○	○	○	検索条件 大文字と小文字を区別
未	bool	FindOptWholeWord	Editor	Editor ○	○	○	検索条件 単語検索
未	bool	FindOptRegexp	Editor	Editor ○	○	○	検索条件 正規表現検索
	int	HitSelLength	Editor	Editor ○	○	×	検索一致文字列長
未	String	TextCode	Editor	Editor ○	○	○	文字コード
未	String	RetCode	Editor	Editor ○	○	○	改行コード
	int	Count	Regex	× ×	○	×	後方参照可能な文字列の数
	int	Pos	Regex	× ×	○	×	ヒットした位置(0基準)
	int	Len	Regex	× ×	○	×	ヒットした文字列の長さ
	int	MatchLen	Regex	× ×	○	×	検索文字列先頭からヒットした文字列終端までの長さ

[注] 未: 現在未実装の意味です。

・メソッド

戻り値	名前	Lua	DM COM	引数	説明
int	CharCount	○	Editor ○	str	文字数カウント str-文字列
int	RowToLines	Editor	Editor ○	Row	Row で指定された行の Lines 上の行インデックスを返します
int	LinesToRow	Editor	× ○	i	Lines 上の i で指定された行に対応する Row を返します。
int	ColToChar	Editor	Editor ○	Row,Col	(Row, Col)位置の、Lines 上での文字インデックスを返します
String	GetLineString	Editor	Editor ×	i	指定行の文字列を取得 i-行番号(Line カウントで指定)
void	SetLineString	Editor	Editor ×	i,str	指定行に文字列を設定 i-行番号(Line カウントで指定), str-文字列
void	BeginUpdate	Editor	Editor ×	なし	エディタ画面の描画を抑制
void	EndUpdate	Editor	Editor ×	なし	エディタ画面の描画を有効

bool	Save	Editor	×	×	fname	編集中のファイルをファイル名 fname で保存する
bool	Load	Editor	×	×	fname	ファイル名 fname のファイルを読み込む
bool	IsMarked	Editor	Editor	×	i	マーク設定されているか否か i-行番号(Line カウントで指定)
void	SetMarked	Editor	Editor	×	i,on	マークを設定/解除する i-行番号, on-true: 設定/false: 解除
void	Find	Editor	Editor	×	なし	検索
void	FindNext	Editor	Editor	×	なし	次を検索
void	FindPrev	Editor	Editor	×	なし	前を検索
void	Replace	Editor	Editor	×	なし	置換
void	HitToSelected	Editor	Editor	×	なし	ヒットした部分を選択状態にする
void	ClearUndo	Editor	Editor	×	なし	Undo バッファをクリア
void	Undo	Editor	Editor	×	なし	元に戻す(Undo)
void	Redo	Editor	Editor	×	なし	やり直し(Redo)
void	Cut	Editor	Editor	×	なし	切り取り
void	Copy	Editor	Editor	×	なし	コピー
void	Paste	Editor	Editor	×	なし	貼り付け
void	Delete	Editor	Editor	×	なし	削除
void	SelectAll	Editor	Editor	×	なし	全てを選択
void	InsertFileName	Editor	Editor	×	なし	ファイル名の挿入
void	SelectWordFromCaret	Editor	Editor	×	なし	キャレット位置の単語を選択
void	SelectTokenFromCaret	Editor	Editor	×	なし	キャレット位置のトークンを選択
void	SelectLineFromCaret	Editor	Editor	×	CrLf	キャレット位置の行を選択 CrLf-true:改行含む /false:改行含まない
void	Wait	○	Time	○	ms	ms-待ち時間(msec で指定)
int	GetTime	○	Time	○	なし	システム時刻(Windows 起動後の経過時間)をミリ秒単位で取得
String	UrlEncode	○	×	×	str	str(文字列)を UTF8 で URL エンコードする
String	UrlDecode	○	×	×	str	UTF8 で URL エンコードされた str(文字列)を sjis にデコードする
String	ClipBoard	○	×	×	str	ｸﾘｯﾌﾟﾎﾞｰﾄﾞに str(文字列)を保存(未指定可)。保存前の内容取得。
void	ShellExecute	○	×	×	F(,P(,D))	Shell 実行。F:操作対象のﾌｧｲﾙ, P:操作のﾊﾟﾗﾒｰﾀ, D:既定のﾃﾞｨﾚｸﾄﾘ
void	ProcessMessages	○	×	×	なし	メッセージキューを処理できるよう, 実行を一時的に停止する
※1	InpPara	○	×	×	※1	複数入力パラメータ Dialog の表示

※2	FileDialog	○	×	×	※2	ファイル選択用Dialogの表示
※3	SelMenu	○	×	×	※3	選択メニュー表示
bool	Match	Regex	×	×	※4	パターンマッチ処理を実行

※1: InpPara(Title, Para1, Val1, [Para2, Val2, ...])

引数

Title : タイトル
Para1 : 項目1名
Val1 : 項目1初期値
Para2 : 項目2名
Val2 : 項目2初期値
...

戻り値

OKか否か, 項目1の入力値, 項目2の入力値, 項目3の入力値, ...

使用例

Luaの場合

```
tbl = {456,"x","y","z"}

ret,a,b,c = InpPara("入力test",
                    '数値' , 123,
                    '文字列', 'abc',
                    'ﾃｰﾌﾞﾙ' , tbl)
if ret==false then
        printf("¥nCancel!!¥n¥n")
end

printf("1: "..a.."¥n")
printf("2: "..b.."¥n")
printf("3: "..c.."¥n")
```

※2: FileDialog([mode, [Title, [InitialDir]]])

引数

mode : 0-OpenDialog, 1-SaveDialog (デフォルト 0)
Title : タイトル(文字列)
InitialDir : Dialogの初期フォルダ

戻り値

FileTbl : 選択(複数)ファイルのテーブル。(キャンセル時は、nil)

使用例

例1 FNameTbl = FileDialog()
例2 FNameTbl = FileDialog(1)
例3 FNameTbl = FileDialog(0, "開くファイルを選択してください")

※3: SelMenu(Title, ColCount, MenuTbl)

引数

Title : タイトル(文字列)
ColCount : 0以下は、横一列で全て表示 / 1以上は、指定数で折り返して表示

MenuTbl : メニュー項目テーブル

戻り値
キャンセル : 0
選択時 : 選択した項目番号(1 以上の数値)

例
```
tbl = {'項目 1','項目 2','項目 3'}                    -- テーブル初期化
ret = SelMenu("MenuTest", 3, tbl)                    -- パラメータ問い合わせ
if ret<=0 then                                  -- 「Cancel」が押された
  printf("¥nCancel!!¥n¥n")
else
  printf("戻り値: "..ret.." ("..tbl[ret]..")¥n")   -- 戻り値の値
end
```

※4: Match(Pattern, Str)
引数
Pattern : パターンマッチ文字列 (正規表現ライブラリ 鬼車のルール参照のこと)
Str : 検索対象となる文字列

戻り値
true : パターンマッチ成功
false : パターンマッチ不成立

例
```
r  = Regex{}
r:Match("(b+)c","aabbccddee")
print('Count   = '..r.Count)
print('Pos     = '..r.Pos)
print('Len     = '..r.Len)
print('MatchLen= '..r.MatchLen)
for I = 1, #r do
  printf("r[%d] = %s¥n", I, r[I])
end
```

結果
```
Count   = 2     ... 後方参照可能な文字列の数(#r と r.Count は同じ値)
Pos     = 3     ... パターンマッチした先頭位置(3 文字目)
Len     = 3     ... パターンマッチした長さ(bbc の 3 文字)
MatchLen= 5     ... 先頭から検索終端までの長さ(aabbc の 5 文字)
r[1] = bbc      ... (b+)c の部分の文字列
r[2] = bb       ... (b+)  の部分の文字列
```

●Script の特別ルール
Script の先頭の 15 行に下記記載をすることで、特殊な効果が得られます。
Title = ○○○ ・・・ メニューにおける Script のタイトルを○○○に設定
Input ・・・ エディタ部で現在選択している領域を、Script へ標準入力として与え

ます。
ただし、現在 Lua, Dmonkey は、標準入力に対応できていません

Input = ○○○ ・・・ エディタ部で現在選択している領域を、○○○というファイルに保存後、Script を実行します。

Output ・・・ Script の標準出力結果で、エディタの現在選択している領域を書き換えます。

Output = ○○○ ・・・ Script 実行後、○○○というファイルの中身で、エディタの現在選択している領域を書き換えます。

Cmd = ○○○ ・・・ Script を実行するためのコマンド名を指定します。

#!○○○ ・・・ Script を実行するためのコマンド名を指定します。
フォルダの区切りは、'¥'以外に、'/'も使えます。
ドライブ名 'C:'は、省略可能です。
本処理は、'/'を全て'¥'と解釈するので注意が必要です。

Para = ○○○ ・・・ Script 実行時に与えるパラメータを指定します。
パラメータには、下記マクロ文字が使用できます。
$P：カレントディレクトリ
$p：カレントファイルのディレクトリ
$D：カレントドライブ
$d：カレントファイルのドライブ
$F：現在編集中のファイル（フルパス）
$f：現在編集中のファイル（ファイル名のみ）
$C：現在編集中のファイル（拡張子を含む,$f と同じ）
$X：現在編集中のファイル（拡張子を含まない）
$Z：Zed の絶対パス(Zed.exe 含む)
$z：Zed ディレクトリ
$$：'$'自身

Opt = ○○○ ・・・ Script 実行時の動作オプションの設定
大文字と小文字の同時指定はできません。大文字が尊重されます。
H ： Script 実行時に、編集中のﾌｧｲﾙがあるフォルダ(Home)に移動
h ： Script 実行時のみ、編集中のﾌｧｲﾙがあるフォルダに一時的に移動
S ： Script 実行前に、編集中のﾌｧｲﾙを強制的に保存(Save)
s ： Script 実行前に、編集中のﾌｧｲﾙを保存するか問い合わせる
例: Opt = HS

ShortCut = ○○○ ・・・ マクロ実行のためのショートカットキーを指定します。
例: ShortCut = Ctrl+Q

・例 1. sort.bat

```
@ECHO OFF
rem Title = 選択領域ソート実行
rem Input
rem Output
rem ShortCut = Shift+Ctrl+S
sort
```

上記、sort.bat の場合、エディタの現在選択している領域が、DOS の sort コマンドの
入力として実行され、その出力結果で、エディタの現在選択している領域を書き換えます。
具体的な動作で書くと、
3

```
  2
  1
```

と書かれた部分を選択後、サブウィンドウ or メニューから、sort を実行すると

```
  1
  2
  3
```

となります。
また、コマンドの実行は、"Shift+Ctrl+S"でも可能です。

・例 2. abc.pl

```
#!/Perl/bin/MSWin32-x86-object/jperl.exe -w
# Para = -K $F
…
```

上記、abc.pl の場合、以下のように実行されます。
C:¥Perl¥bin¥MSWin32-x86-object¥jperl.exe -w abc.pl -K 現在編集中のファイル

Zed の入手先

Kuze's soft Down Load room
http://www.hi-ho.ne.jp/kuze/tool.htm

Zed のサポート

晴れときどき更新中 掲示板：@nifty レンタル掲示板
http://kuze.white.coocan.jp/

著者の Zed 関連活動

Zed Watching - TSNETWiki on TextWorld
http://text.world.coocan.jp/TSNET/?Zed%20Watching

（投稿日 2012 年 10 月 7 日）

iUI 入門 - 更新日記モバイル版

jscripter

はじめに

iUI はスマートフォンやハイエンドのモバイルデバイス用の Web フレームワークである。2007 年に Joe Hewitt によって、彼のブログに発表された。

> iUI - Mobile web framework for high-end devices
> http://www.iui-js.org/
>
> Introducing iUI - Joe Hewitt
> http://joehewitt.com/2007/07/11/introducing-iui

著者が iUI を知ったのは 2011 年に Hewitt が Facebook を辞めたのが話題になった時のことである。その頃は、おそらくリリース 0.31 だったはずで、結構実用的という感触を得ていた。今年調べると、0.40 が出ていたので、もう少し詳しく調べた。

> Joe Hewitt (programmer) - Wikipedia, the free encyclopedia
> http://en.wikipedia.org/wiki/Joe_Hewitt_(programmer)
>
> 更新日記 - 日曜プログラマのひとりごと - 更新日記を iPhone へ(2011/05/08)
> http://homepage1.nifty.com/kazuf/renewal_2011_05.html#iphone_1304828378
>
> iui - Web UI Framework for mobile devices
> - iOS, Android, Palm, and others - Google Project Hosting
> http://code.google.com/p/iui/

本稿では、著者が新たに構築を始めた更新日記モバイル版を実例にして iUI でどのようなことが実現できるのかを示したい。作るにあたって iUI の新しいサイト(http://www.iui-js.org/)のドキュメントを参考にした。なお、ライセンスはコードとコンテンツの両方について Google Code のサイトで明確に示されているので参照すること。

更新日記モバイル版の構成と内容

```
renewal_iui.html ---+--- renewal_iui_2012_10.html
                    |
                    +--- renewal_iui_2012_09.html
                    |
                   ...
                    |
                    +--- renewal_iui_2011_12.html
                   ...
```

現段階では、renewal_iui.html で年の選択に続いて月の選択をし、各月の一塊のデータをまとめた renewal_iui_yyyy_mm.html でカテゴリの選択、記事の選択、記事の表示を行う仕組みを試している。

[renewal_iui.html]

```
<!DOCTYPE html>
<html>
<head>
  <title>更新日記: 日曜プログラマのひとりごと</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
user-scalable=0"/>
  <link rel="icon" type="image/png" href="iui/iui-favicon.png">
  <link rel="apple-touch-icon" href="iui/iui-logo-touch-icon.png" />
  <link rel="stylesheet" href="iui/iui.css" type="text/css" />
  <link rel="stylesheet" title="Default" href="iui/t/default/default-theme.css"
type="text/css"/>
  <link rel="stylesheet" href="css/iui-panel-list.css" type="text/css" />
   <script type="application/x-javascript" src="iui/iui.js"></script>
</head>
<body>
<!-- start of variable data -->
    <div class="toolbar">
        <h1 id="pageTitle">更新日記: 日曜プログラマのひとりごと</h1>
        <a id="backButton" class="button" href="#"></a>
    </div>
<div id="home" class="panel" selected="true">
  <fieldset>
        <p class="normalText"><b>更新日記にようこそ。日曜プログラマのひとりごとです。現在、
テスト中(α版)ですので、データは不完全であり、かつ表示の仕方も変化しています。</b></p>
  </fieldset>
    <ul id="home" title="日曜プログラマのひとりごと" selected="true">
        <li><a href="#2001">2001年</a></li>
        <li><a href="#2002">2002年</a></li>
        <li><a href="#2003">2003年</a></li>
        <li><a href="#2004">2004年</a></li>
        <li><a href="#2005">2005年</a></li>
        <li><a href="#2006">2006年</a></li>
        <li><a href="#2007">2007年</a></li>
        <li><a href="#2008">2008年</a></li>
        <li><a href="#2009">2009年</a></li>
        <li><a href="#2010">2010年</a></li>
        <li><a href="#2011">2011年</a></li>
        <li><a href="#2012">2012年</a></li>
    </ul>
  <h2>更新日記モバイル版について</h2>
  <fieldset>
        <p class="normalText">iUIを使用した更新日記ブラウザ。</p>
        <p>powered by <a target="_blank" href="http://code.google.com/p/iui/">iui - Web UI
Framework for mobile devices - iOS, Android, Palm, and others - Google Project
Hosting</a>.</p>
  </fieldset>
```

```
  </div>
    <ul id="2001" title="2001年">
      <li><a target="_blank" href="renewal_iui_2001_01.html">1月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_02.html">2月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_03.html">3月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_04.html">4月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_05.html">5月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_06.html">6月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_07.html">7月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_08.html">8月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_09.html">9月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_10.html">10月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_11.html">11月</a></li>
      <li><a target="_blank" href="renewal_iui_2001_12.html">12月</a></li>
    </ul>
.....
    <ul id="2012" title="2012年">
      <li><a target="_blank" href="renewal_iui_2012_01.html">1月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_02.html">2月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_03.html">3月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_04.html">4月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_05.html">5月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_06.html">6月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_07.html">7月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_08.html">8月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_09.html">9月</a></li>
      <li><a target="_blank" href="renewal_iui_2012_10.html">10月</a></li>
    </ul>

</body>
</html>
```

図1. 更新日記通常版の Safari 表示画面

著者の更新日記のページを iPod touch(第3世代、iOS5.1)の Safari で表示すると左図のようになって、見辛い。元々縦長のデザインで小さな画面用の構成を取っていたので、iPad であれば問題ないが、さすがに iPod touch ではむずかしい。

通常版(renewal.html)を iUI の Web フレームワークに移植した renewal_iui.html は次のように表示される。

図 2. renewal_iui.html の初期画面

renewal_iui.html の div タグの panel の class の範囲が表示されている。

fieldset タグを置くと枠の中にコメントなどを表示できる。

年ごとの月へのリンクリストが表示される。ハイパーリンクはアンカーで示し、各月の ul タグの id と対応関係がある。

図 3. renewal_iui.html の月選択画面

各月のハイパーリンクには、次に示す各月のカテゴリと記事を格納した HTML へのリンクを設定している。

[renewal_iui_2012_10.html]

```
<!DOCTYPE html>
<html manifest="renewal.manifest">
<head>
<title>iUI 更新日記: 2012 年 10 月</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
   <meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0;
user-scalable=0;"/>
   <link rel="icon" type="image/png" href="iui/iui-favicon.png">
   <link rel="apple-touch-icon" href="iui/iui-logo-touch-icon.png" />
   <link rel="stylesheet" href="iui/iui.css" type="text/css" />
   <link rel="stylesheet" href="iui/t/default/default-theme.css" type="text/css"/>
   <script type="application/x-javascript" src="iui/iui.js"></script>
</head>

<body>
    <div class="toolbar">
        <h1 id="pageTitle"></h1>
        <a id="backButton" class="button" href="#"></a>
    </div>
<!-- start of variable data -->
       <ul id="home" title="2012年10月" selected="true">
       <li><a href="#apple">Apple</a></li>
       <li><a href="#prog">プログラミング言語</a></li>
       <li><a href="#techip">知的生産の技術</a></li>
       </ul>
       </ul>
       <ul id="apple" title="Apple">
       <li><a href="#apple_1349096845">2012-10-01 22:07:25 - デザインを超えるもの</a></li>
       <li><a href="#apple_1349440998">2012-10-05 21:43:18 - まだ書かれていない歴史のページ<
/a></li>
       </ul>
       <ul id="prog" title="プログラミング言語">
       <li><a href="#prog_1349274254">2012-10-03 23:24:14 - Joe HewittのUp</a></li>
       </ul>
       <ul id="techip" title="知的生産の技術">
       <li><a href="#techip_1349098002">2012-10-01 22:26:42 - 手帳とデジタル</a></li>
       </ul>
.....
       <div id="prog_1349274254" class="panel" title="2012-10-03 23:24:14">
             <h2>[プログラミング言語] Joe HewittのUp<h2>
             <p><a href="https://github.com/joehewitt/up">joehewitt/up - GitHub</a>ネタ。二
日前か。TwitterやFacebook、ホームページも更新が停止しているので、ググって調べた成果
^^)/</p>
<p>Facebookを昨年辞めたjoehewittがようやく動き始めたようだ。Facebook for iPhoneの作者とし
て有名だし、僕が最近触っているiUIを立ち上げたのもHewitt、Firebugや初期のFirefoxにも関わっ
ていた。<a href="https://addons.mozilla.org/en-us/firefox/addon/firebug/">Firebug :: Add-
ons for Firefox</a>は今でもメインプレーヤーなのか。</p>
<p><a href="http://joehewitt.com/2007/07/01/firebug-for-iphone" target="_webapp">Firebug
for iPhone - Joe Hewitt</a>を見ても、動き出したUpを見ても、Pythonがお好きなようだ。</p>

       </div>
.....
</body>
</html>
```

図 4. 各月のカテゴリ選択画面

図 5. カテゴリの記事選択画面

図 6. 記事表示画面

各記事に含まれる外部へのリンクはタップで表示可能だが、更新日記内の記事には直接飛べないのが、現状最大の問題で、カテゴリ選択画面が表示されてしまう。

HTML の記述と表示画面の対応はわかりやすいので、これ以上、くどくどと説明はしない。どのような HTML を生成すればよいかさえわかれば、様々に応用が効くだろう。

iUI - Getting Started

http://www.iui-js.org/documentation/latest/getting-started.html

上記の iUI のドキュメンテーションのページがもっと様々な表現への示唆を与えてくれるだろう。是非、訪れること。

おわりに

モバイルデバイス用の Web 表現が求められている。iUI のような Web フレームワークが公開されて使うことができるのは大変ありがたいことだ。さらには、モバイルデバイス用の表現の工夫における革新がデスクトップの表現をも変えていくのではと予感している。それを支えるのは言うまでもなく JavaScript である。

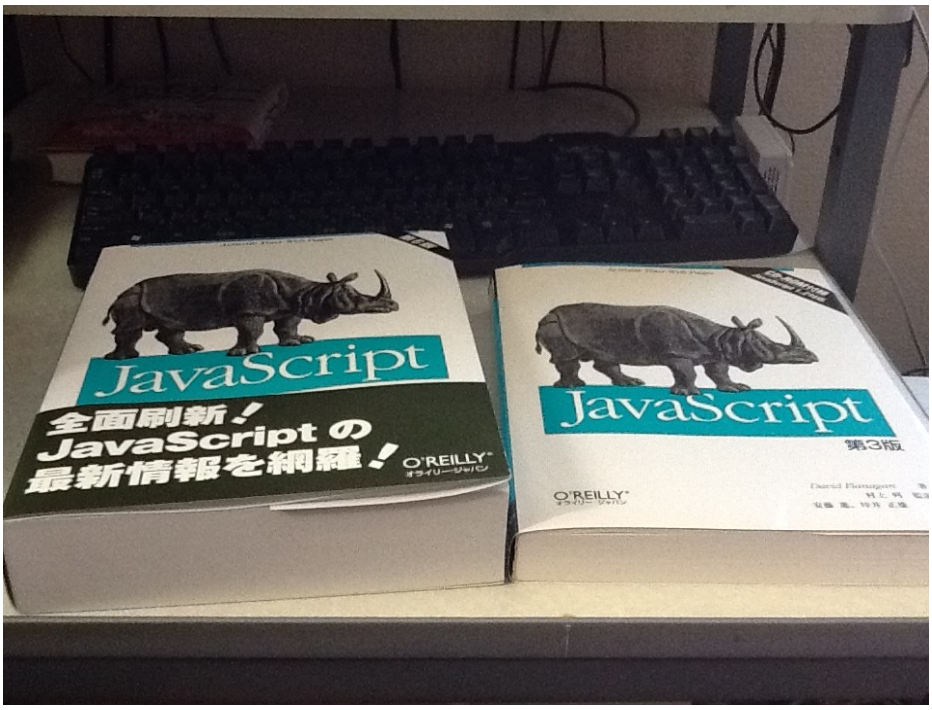

図 7. JavaScript 第 3 版と第 6 版

2000 年の JavaScript 第 3 版が古くなったのを実感して、今年(2012 年)出た第 6 版を購入した。ボリュームも倍になった。もちろん、LiveScript として登場した時から知らぬわけではない存在であり、これまでも触ってこなかったわけでもないが、そろそろ本格的に取り組まねばならぬと遅まきながら考える昨今である。

(投稿日 2012 年 10 月 7 日)

編集後記

jscripter

ようやく、編集後記に辿り着いた。いつもホッとする。好きなことを書いて終えよう。

以前は、TSNET スクリプト通信は一冊だけ印刷していたのだが、インクジェットの印刷コストが高過ぎるのと、iPad2 を購入したので止めてしまった。しかしながら、紙媒体も実際のところ便利なのだ。もっと印刷コストを下げて欲しいと思う。紙媒体の印刷においても何か革命的なことはできないのだろうか。いつもそう思う。

さて、それはともかく、IT の変革は続いていく。iPad mini、Windows 8/METRO、Firefox OS など、目白押しだ。最近、テレビのコマーシャルを見ていると、パナソニックの家電分野での攻勢が目立つ。ほーっ、がんばっているじゃないと思う。

「ビエラ・コネクト」スペシャルサイト | プラズマテレビ/液晶テレビ ビエラ | Panasonic
http://panasonic.jp/viera/viera_connect/top.html

でも、このサイトも複雑過ぎる。焦点が合わないというか。もっと凄いことができるというような現実歪曲空間を生み出さないとインパクトがない。

オーディオ・ビデオとテレビとコンピュータを融合させる必要がある。それはまだアップルも完全にできていない。

オーディオ&ビジュアル製品情報＞DS-A5(B)
http://www.jp.onkyo.com/audiovisual/option/dsa5/

最近、これこれと思った製品。AirPlay とオーディオ機器を結びつける。出ないかなあと思っていたら、さすが、PC-Audio の先駆者だね。むずかしいのは、環境を整えるのに様々な部分にコストが掛かること。ユーザーはゆっくりしか進めない。

Zazel さんのツイートで、マインドストーム NXT への興味を再燃させたのだが、爆熱コンピュータを抱える身では、先送りするしかないのが結論^^;)

レゴ マインドストーム公式サイト
http://www.legoeducation.jp/mindstorms/

世界の経済や政治は混迷しつつも、IT・コンピューティングの発展だけは加速しつつある。すごい世の中になったものだ。

(投稿日 2012 年 10 月 7 日)

ＴＳＮＥＴスクリプト通信

ISSN 1884-2798　出版地: 広島市

2012 年 10 月　8 日　　5.1.000 版
2012 年 10 月　8 日刊行　　5.1.001 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5巻第1号(通算第17号)
発行: TSC編集委員会 発行日: 2012年 10月 8日
ISSN: 1884-2798 出版地: 広島市 創刊: 2008年5月7日