Ruby/CHISEの実装, IDSによる字形構造分析の試み

独立行政法人産業技術総合研究所 特別研究員 江渡 浩一郎 k-eto@aist.go.jp

まずはデモ。

```
require 'chise'
include CHISE
p "字"                          #"字"
p "字".ucs                      #23383
p "字".total_strokes            #6
p "字".inspect_all              #いろいろでてくる。
p "字".ideograph_daikanwa       #6942       #大漢和番号6942である。
```

このように実行することができる。
つまり文字そのものが自分の持つ属性を知っていて、そこにメソッドとしてアクセスできる。

Character class

クラスの拡張ではなくCharacter classに実装されている。

文字列(String)の長さが、UTF-8的な意味で一文字の場合のみ、内部で自動的にCharacterに変更して扱う。
そのため、String自体を拡張しているように見える。一文字以外のときはエラーとなる。

```
エラー
```

オブジェクトを生成し、それを直接用いることもできる。

```
                                で、その文字列からCharacter classのインスタンスを生成。
char = Character.get("字")      #上記とまったく同じ動作を行う。
```

パターンを用いているため、同じ字の場合は同じインスタンスとなる。

実体参照

```
                                実体参照化する
p "&#x5b57;".de_er              #"字"              #実体参照を解除
p "&J90-3B7A;".de_er            #JISX0208-1990におけるコードポイントを用いた実体参照を解除
p "&M-06942;".de_er             #大漢和番号を用いた実体参照を解除
```

このように、実体参照を扱うことができる。様々なCCSにおいて実体参照を解除することができる。

の拡張

String#each_character の一文字毎にイテレータを実行する。引数としてCharacterがはいる。
同様にmapした結果の文字列を返す。

属性の定義

```
木".mydepth = 1
"林".mydepth = 2
"森".mydepth = 3
```

データベースに保存されるので、プログラム終了後も属性は保存される。

```

```

このようにして、自分独自の必要な属性を作り、その属性に基づいたプログラムを書くことができる。

IDSによる字形分解・合成ができる。

```
                                ⿸宀子"     #分解する
p "⿸宀子".compose               #"字"       #合成する
p "鬱".decompose                #"木缶木冖鬯彡"          #難しい字も分解できます。
p "鬱".decompose_all            #これ以上分解できないところまで再帰的に分解する。
p "木缶".find                   #二つの部品を含む漢字をリストアップする。
```

字形構造は内部的にはできるだけ集約した形で持っている。

による字形構造分析

このような機能を用いると、漢字の字形内部の構造をデータとして扱うことができる。これを元に字形構造の分析を行うことができる。たとえば、左右に分割される漢字の割合、上下に分割される漢字の割合などを、それぞれ各々の文字集合ごとに統計的に処理することができる。このようにして、字形構造分析を効率的に行うための基盤が構築された。この基盤を元に、より詳細な字形構造分析を今後行う予定である。

不足している機能

との相互変換だけは対応しているが、それ以外は対応していない。外部ライブラリーをヘルパーとして用いて対応することを考えている。

```
http://eto.com/2003/ruby/
```

ここから入手できる。現時点では、XEmacs CHISEにおいてビルドされた文字データベースが必要である。
Windows, Linuxでも同様に動作する。開発は主にWindows上でおこなった。