正規表現を活用しよう

- 書式チェックだけでは勿体ない。文字列加工や引数チェックも可能ですよ
- 「使えない!」とならないために

Ognac

自己紹介

- Ognac
- 　アマチュア無線Call Sign
- 　JE3OGN(じいさんおじん)
  AutoMaticComputer (Eniac/Tosbac..)
- 関西でフリーターデベロッパー(MS系)
- 汎用機コボル(vm/MVS･S/360)で業界入り
- VBシンパだったが、気がつけばC#派に

- 正規表現-----何が正規？
- 　正規分布- Normal Distribution
- 　RegexもNormalも正規は変
- ・「正式」「正規の」「公式」と誤解をしている人も

- OS環境/:ベンダー環境で方言が強い
- 環境が異なれば正規表現の書式動作異

- 正規表現エンジンは3種類
- 1. 非決定性有限オートマトン (NFA) エンジン
- 　　バックトラッキング正規表現検索エンジン
- 2. 正規表現決定性有限オートマトン (DFA) エンジン
- 　　高速であるが制限が多い
- 3. POSIX NFA
- 　　標準化されているが低速な
-
- .NET はFNA式

使い道

テキストボックスの入力値の書式チェック

テキストデータの検索

テキストデータの加工

与えられた、指示、命令の引数解析

- 単なる道具です。万能薬では有りません
- ｱﾙﾌｧﾍﾞｯﾄ圏内の道具なので日本文と合わない部分もある。
- 分かち書きのない日本文には合わないオプションもある
- 文末や行末が¥cr¥lf でなく¥nしか認識してくれない
- 単語の区切り(¥w ¥b など)が日本文には適さない

制約は多いが、それ以上のメリットがある

一般化しないのは説明が独特だから?

- オートマトンがベースなので不可避な表現だが
- .* 任意の文字の0回以上の繰り返し
- .* 任意の文字の1回以上の繰り返し
- (朝|夕)?日　:「朝」か「夕」が0回または1回の「日」

- 一つの記号に複数の意味がある
- (?=大阪市).*区(.*?)町
- 　　左の()は肯定先読みで?はキャプチャしない
- 　　右の()はグループ化で内容を取得する
- 　　右の? は最短検索
- 混乱しますよね。

- 数字3桁： ¥d{3}
- 数字3桁　^¥d{3}$
- 数字1～3桁^¥d{1,3}$
- 100 or 200 ^(100|200)$
- 100と200以外の3桁　^(?!100)(?!200)¥d{3}$

- 単純な郵便番号 ^¥d{3}-¥d{4}$
- 000-0000,999-9999を弾く
- 　　^(?!000-0000)(?!999-9999)¥d{3}-¥d{4}$"

パスワードの有効性Check

- 8桁以上の数字と小文字と大文字混じり
- ^(?=.*¥d)(?=.*[a-z])(?=.*[A-Z])¥w{8,}$

- ^[¥d-[3-4]]+$
- 0から9の数字で　3,4を弾く

カンマ区切りのCSVの切り出し

- a, bb, cc , ddを切り出す
- ¥b(?<fact>.*), 期待と違う
- ¥b(?<fact>.*?)(?=(,|$)) 巧くいった

- 文字列が”xxxxx” , ‘kkkkkkk”で文字列として”,”がある
- a, 'B1,B2,B3' , ¥"Z1,Z2,Z3¥" , cc , dd

- ¥s*(?<q1>[""']?)(?<fact>.*?)(?(q1)¥k<q1>)¥s*(?=(,|$))

日付の置換/住所のXML化

- 12-2-1999 10/23/2001 4/5/2001
- ¥b(?<mm>¥d{1,2})(?<sep>(/|-))(?<dd>¥d{1,2})¥k<sep>(?<yy>(¥d{4}|¥d{2}))¥b ¥n ${yy}${sep}${mm}${sep}${dd}

- 字面の置換なので限度がある

- 01101,"064 ","0640941","ﾎｯｶｲﾄﾞｳ","ｻｯﾎﾟﾛｼﾁｭｳｵｳｸ","ｱｻﾋｶﾞｵｶ","北海道","札幌市中央区","旭ケ丘",0,0,1,0,0,0

郵便番号CSVのXML化

- ¥¥s*(?<q1>[¥"']?)(?<d1>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d2>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<ZIP>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d3>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d4>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d5>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<県名>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<市名>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<町域名>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d6>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d7>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d8>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<d9>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<da>.*?)(?(q1)¥¥k<q1>)¥¥s*,¥¥s*(?<q1>[¥"']?)(?<db>.*?)(?(q1)¥¥k<q1>)¥¥s*$

```
var s = re.Replace(line,      delegate(Match mc)
                        {
                          //cnt += 1;  // 広域変数でつかえる
                            return "<zip>" + mc.Groups["ZIP"] + "</zip>" //<県名>${県名}</県名><市名>${市名}</市名><町域名>${町域名}</町域名>¥n";
                                + "<県名>" + mc.Groups["県名"] + "</県名>"
                                + "<市名>" + mc.Groups["市名"] + "</市名>"
                                + "<町域名>" + mc.Groups["町域名"] + "</町域名>"
                                + System.Environment.NewLine
                                ;
                        }
);
```

区名の抜き出し

- 大阪市と神戸市の区を抽出
- ^.*(?=(大阪|神戸)市).*区.*$

- ・政令都市以外の区
- ^.*(?<!(札幌|仙台|さいたま|千葉|横浜|川崎|新潟|静岡|浜松|名古屋|京都|大阪|堺|神戸|広島|北九州|福岡))市.*区.*$

- ^.*(?<!(札幌|仙台|さいたま|千葉|横浜|川崎|新潟|静岡|浜松|名古屋|京都|大阪|堺|神戸|広島|北九州|福岡))市.*[^一二三四五六七八九123456789]+区.*$

便利か知らんけど遅いのでは？

- 遅いか早いかは、スキルしだい
- 一二万件のを素読みしてみる
- while ((line = sr.ReadLine()) != null)
- {
- continue;
- }

- 大阪市と神戸市のデータをC#で抽出
- if (line.IndexOf(¥"大阪市¥") > -1 || line.IndexOf(¥"神戸市¥") > -1)

- 正規表現で抽出
- ^.*(大阪市|神戸市).*$
- 下手な時
- .*(大阪市|神戸市).*

構文解析もどき

- ネストされた引数の分離
- この式のabc( ) の引数や.　F1()の引数を抜き出したい
- ABC(KBC,F1(fa1,fb2) , G2(g1,g2,g3) , H3(K(2),J(3,4)))

- (?<TITLE>[^¥(¥)]*)(((?'Open'¥()[^¥(¥)]*)+((?'Close-Open'¥))[^¥(¥)]*?)+)*(?(Open)(?!))

まとめ

- Regexは万能ではない道具
- 慣れるまでが大変
- 処理系の方言に注意
- すべての漢字は[一-龠]等Unicode
- [一-熙] SJISなどコード体系に依存するので注意が必要
- ひらがな、カタカナなどのグループ指定
- は両端に無文字が含まれる
- 学習は慣れるしかないが、式が作れるようになると、言語で判別するより早く実装できる。