テキスト処理 第8回 (2008-06-10)

田中哲
産業技術総合研究所
情報技術研究部門
akr@isc.senshu-u.ac.jp
http://staff.aist.go.jp/tanaka-akira/textprocess-2008/

今日の内容

- 前回のレポートの説明
- キャプチャの説明
- MatchData
- キャプチャの実装
- 任意の一文字
- レポート

キャプチャ

- マッチ全体ではなく、その一部を得る
- パターン中の丸括弧に対応するところを得る
- 特殊変数 $1, $2, $3, ... で対応する箇所を参照する

```
p /(.*)=(.*)/ =~ "favorite=banana"   #=> 0
p $1                                 #=> "favorite"
p $2                                 #=> "banana"
```

キャプチャの使用例 (1)

- URL の scheme を取り出す
  - http, ftp, mailto, ...
- /¥A([^:]*):/ =~ "http://www.senshu-u.ac.jp/"
  p $1   #=> "http"
- /¥A([^:]*):/ =~ "mailto:foo@example.org"
  p $1   #=> "mailto"

キャプチャの使用例 (2)

- カンマ区切りの表を解析する

```
name = nil
max = 0
ARGF.each {|line|
  if /([^,]*),([^,]*)/ =~ line
    if max < $2.to_i
      name = $1
      max = $2.to_i
    end
  end
}
p [name, max]
```

- 各行は「名前,点数」
- 点数が最大の名前を表示

キャプチャの使用例 (3)

- Ruby プログラムから定義しているメソッド名を取り出す (完璧ではない)

```
ARGF.each {|line|
  if /def  *([A-Za-z0-9_]*)/ =~ line
    puts $1
  end
}
```

括弧を通らない場合

- 全体がマッチしても、括弧の部分を通ってなければ nil になる
- /(a)|(b)|(c)/ =~ "b"
  p $1　　#=> nil
  p $2　　#=> "b"
  p $3　　#=> nil

括弧のネストと番号

- 括弧の番号は、左括弧の位置で決まる
- re = /a(b(c)d(e)f(g(h(i)j)k(l)m)n)o/

```
#        1 2    3    4 5 6       7
re =~ "abcdefghijklmno"
p $1     #=> "bcdefghijklmn"
p $2     #=> "c"
p $3     #=> "e"
p $4     #=> "ghijklm"
p $5     #=> "hij"
p $6     #=> "i"
p $7     #=> "l"
```

shy group : (?:...)

- (?:...) はグループ化はするがキャプチャはしない
- http もしくは ftp URL の scheme を得る

```
/¥A((?:ht|f)tp):/ =~ "http://www.senshu-u.ac.jp/"
p $1    #=> "http"
p $2    #=> nil                キャプチャされてない

/¥A((ht|f)tp):/ =~ "http://www.senshu-u.ac.jp/"
p $1    #=> "http"
p $2    #=> "ht"               キャプチャされてる
```

MatchData

- MatchData オブジェクトはマッチ情報を保持する
- マッチに成功した後、特殊変数 $~ で参照できる
- オブジェクトなので変数に入れたりできる
- p $~ #=> nil
  /¥A((ht|f)tp):/ =~ "http://www.senshu-u.ac.jp/"
  m = $~
  p m[1] #=> "http" $1 と同じ
  p m[2] #=> "ht" $2 と同じ

特殊変数 $1, $2, ...

- $1, $2, $3, ... は $~[1], $~[2], $~[3], ... に対応
- $~.begin(1) は $1 が始まる位置
- $~.end(1) は $1 が終わる位置

```
/(a*)(b*)(c*)/ =~ "abbccc"
m = $~
p m[2]          #=> "bb"
p m.begin(2)    #=> 1
p m.end(2)      #=> 3
```

MatchData (2)

- MatchData には他にもいろいろ機能がある
- p /bb*/ =~ "abbccc"　　#=> 1
  m = $~
  p m.pre_match　#=> "a"　　マッチ前の文字列
  p m[0]　　#=> "bb"　　マッチした文字列
  p m.post_match　#=> "ccc"　　マッチ後の文字列
  p m.begin(0)　　#=> 1　　マッチ開始位置
  p m.end(0)　　#=> 3　　マッチ終了位置
- $~[0] などの 0 はマッチ全体を意味する

特殊変数 $&

- $& は $~[0] とほぼ同じ
- $~ が nil のときには $& も nil
- /bb*/ =~ "abbccc"
  p $&　　#=> "bb"
  $~ = nil
  p $&　　#=> nil

named capture

- 番号は扱いにくい
  - パターンを変更すると番号がずれる
  - 大きなパターンでは数えるのが大変
- Ruby 1.9 (開発版) では名前をつけられる

- p /(?<key>.*)=(?<val>.*)/ =~ "favorite=apple"
  #=> 0
  p $~[:key]   #=> "favorite"
  p $~[:val]   #=> "apple"

- (?<name>pat) にマッチしたものは $~[:name] で取り出せる

正規表現エンジンの拡張: キャプチャ

- 番号をつけるのは面倒なので named capture を実装する
- [:capture, 名前, 正規表現の配列表現]

- キャプチャを拡張した try の実行例:
- try([:capture, :n, "a"],
  ["a"], 0, {}) {|pos, md|
  p pos #=> 1
  p md #=> {:n=>0...1}
  }
- /(?<n>a)/ =~ "a" に対応する

try のキャプチャ拡張

- MatchData ではなく、Hash で名前と場所の対応を保持する
- try の引数とブロック引数に Hash を加える
- Hash の鍵はキャプチャの名前のシンボル
- Hash の値は範囲を表現する Range

Hash

- 整数以外でもアクセスできる Array みたいなもの
- 鍵(key) と値(value) の対応を記録
- {key1=>val1, key2=>val2, ...} で表現
- 今回はシンボルを鍵として使う
- ```
  h = {}              # 空ハッシュ
  h[:tea] = 140
  h[:water] = 120
  p h                 #=> {:tea=>140, :water=>120}
  p h[:tea]       #=> 140
  p h[:water]   #=> 120
  ```
  ```

Range

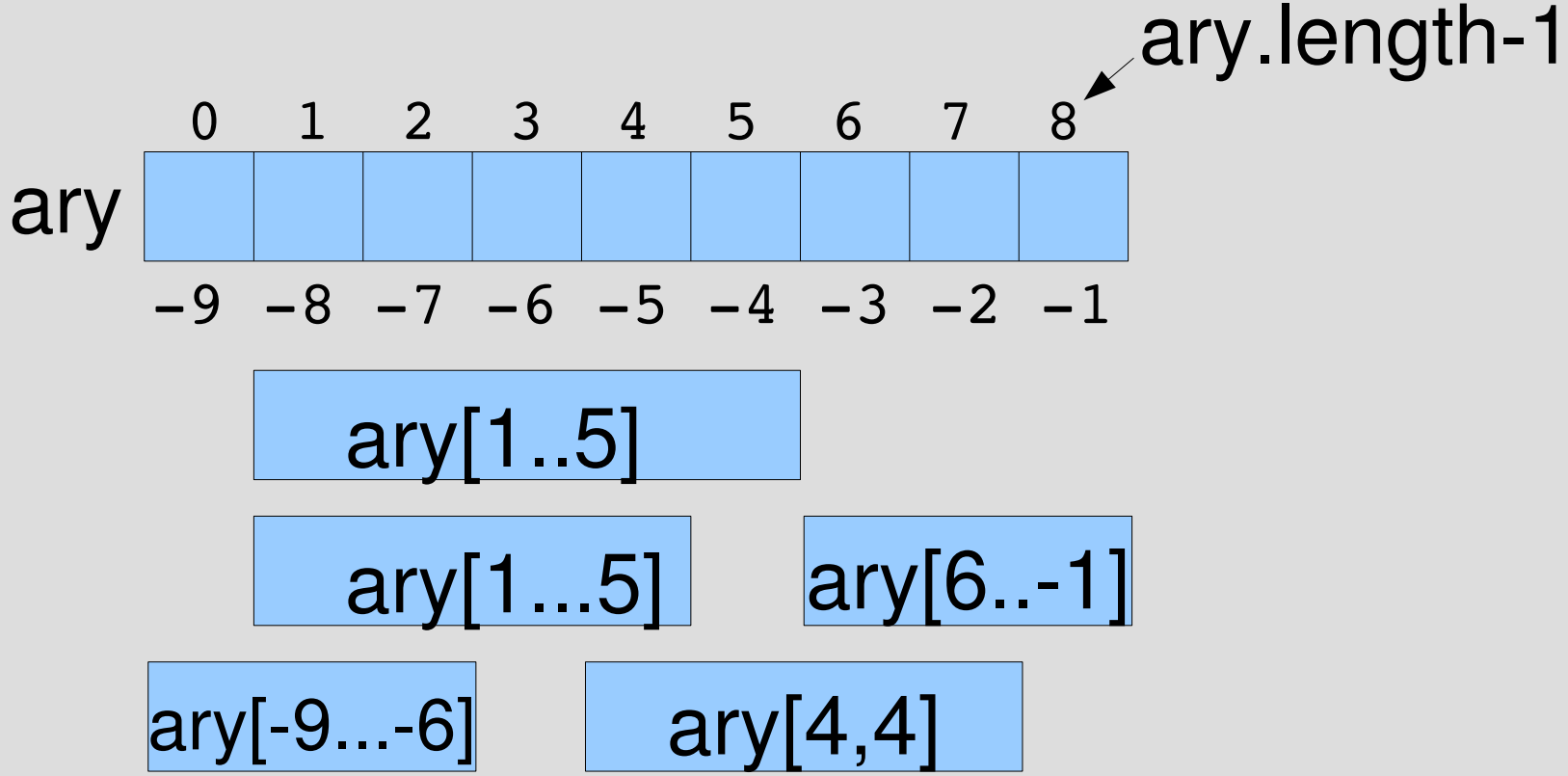
- 範囲を表すオブジェクト
- 3..7 とか 2...5 とか
- 点がふたつのは終端を含む
- 点がみっつのは終端を含まない
- r = 2...5

```
p r.begin                  #=> 2
p r.end                    #=> 5
p r.exclude_end?           #=> true
p (2..5).exclude_end?      #=> false
```

- try では点がみっつのものを使う
- 部分配列を得るなどに使える

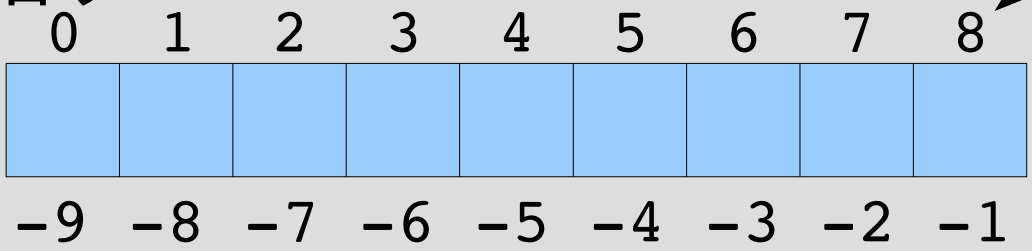
部分配列

- ary[pos1..pos2]　pos1からpos2までの部分配列
- ary[pos1...pos2]　pos1からpos2。pos2 は含まない
- ary[pos,len]　posから長さlen



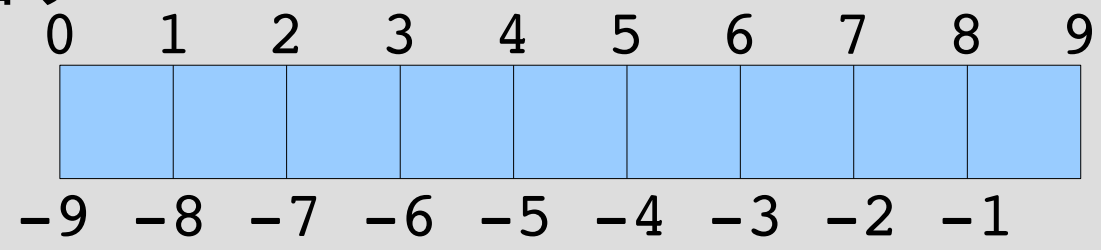
要素を指すか、間を指すか

要素を指す



- a[s..e] に適切

間を指す



- a[s...e] に適切
- 負で最後を指せない

単に考え方の問題
つじつまが合えば、どちらで考えても良い

キャプチャの表現

- [:capture, name, exp] で(名前付)キャプチャを表現
- name はシンボル
- exp は正規表現の配列表現
- Ruby 1.9 の (?<name>exp) に対応する

キャプチャ対応 try

- try(exp, seq, pos, md) {|pos2, md2| ... }
- 以前の try に md, md2 を追加
- md, md2 はキャプチャされた名前から範囲へのハッシュ
- 範囲は s...e という Range で表現
- md にはその try の呼び出しまでに行ったキャプチャの情報を渡す
- md2 は pos から pos2 までのマッチに含まれるキャプチャを md に加えたものになる

try の例

- try([:capture, :n, "a"],
  ["a"], 0, {}) {|pos, md|
  p pos #=> 1
  p md #=> {:n=>0...1}
  }

try の実装

- def try(re, seq, pos, md, &b)
  if re.respond_to? :to_str
  [redacted]
  else
  case re[0]
  [redacted]
  end
  end
  end

単一文字

- ```
  if re.respond_to? :to_str
    yield pos+1, md if str[pos] == re
  ```

- 渡された md をそのまま yield する
- 単一文字のマッチで新しいキャプチャは起きない
  ```

空集合

- when :empset
  # nothing to do

- 変更無し
- マッチしないならキャプチャも何もない

空文字列

- when :empstr
  yield pos, md

- md も yield する
- 空文字列では新しいキャプチャは発生しない

連接 (2引数)

- ```
  when :cat
    try_cat(re, str, pos, md, &b)
  ```

- ```
  def try_cat2(re, str, pos, md, &b)
    try(re[1], str, pos, md) {|pos2, md2|
      try(re[2], str, pos2, md2, &b)
    }
  end
  ```

- md を try(re[1]) に渡して md2 を受け取る
- md2 を try(re[2]) に渡す
- re[1] と re[2] で発生するキャプチャ両方が結果

選択 (2引数)

- when :alt
  try_alt(re, str, pos, md, &b)

- def try_alt2(re, str, pos, md, &b)
  try(re[1], str, pos, md, &b)
  try(re[2], str, pos, md, &b)
  end

- md を try(re[1]) に渡す
- try(re[2]) にも渡す
- re[1] のマッチ時には re[1] でのキャプチャのみ
- re[2] のマッチ時には re[2] でのキャプチャのみ

繰り返し

- ```
  when :rep
    try_rep(re, str, pos, md, &b)
  ```

- ```
  def try_rep(re, str, pos, md, &b)
    try(re[1], str, pos, md) {|pos2, md2|
      try(re, str, pos2, md2, &b)
    }
    yield pos, md
  end
  ```
- md を try(re[1]) に渡して md2 を受け取る
- md2 を try(re) に渡す
- re[1] と re 両方のキャプチャが結果

キャプチャの実装 (1)

- when :capture
  try_capture(re, str, pos, md, &b)
- :capture だったら try_capture で処理する

キャプチャの実装 (2)

- re は [:capture, name, r]
- def try_capture(re, str, pos, md, &b)
  ```
    name = re[1]
    r = re[2]
    try(r, str, pos, md) {|pos2, md2|
      md3 = md2.dup          # ハッシュをコピー
      md3[name] = pos...pos2 # 場所を記録
      yield pos2, md3
    }
  end
  ```
- マッチしたらキャプチャの情報をを追加した md3 を yield する

Hash#dup

- ハッシュのコピーをつくる
- h = {:a => 1}
  h2 = h.dup
  h2[:b] = 2
  p h2　　#=> {:b=>2, :a=>1}
  p h　　#=> {:a=>1}　元のハッシュはそのまま

try_capture の Hash#dup

- def try_capture(re, str, pos, md, &b)
  name = re[1]
  r = re[2]
  try(r, str, pos, md) {|pos2, md2|
  md3 = md2.dup　　# ハッシュをコピー
  md3[name] = pos...pos2 # 場所を記録
  yield pos2, md3
  }
  end

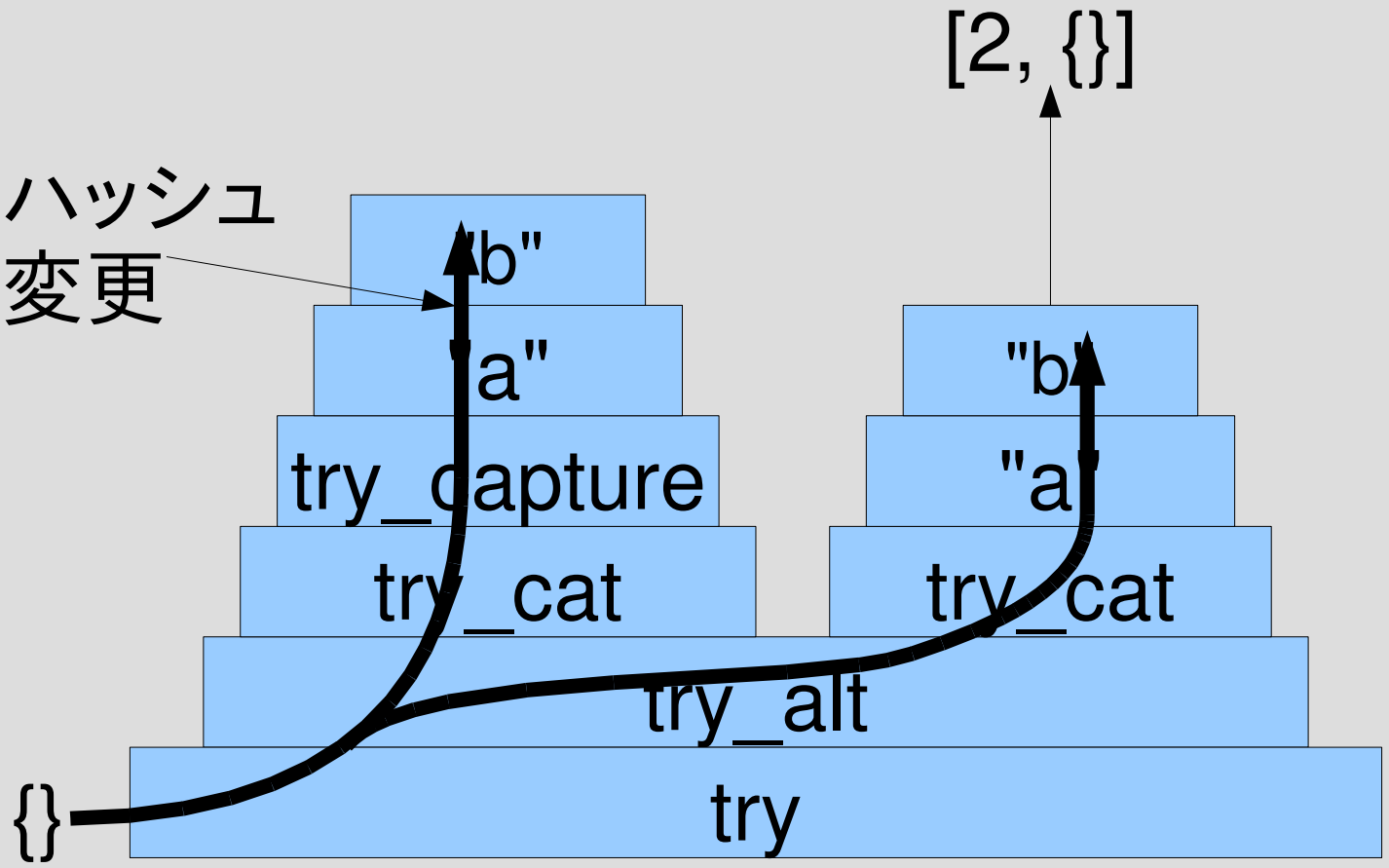
コピーの必要性

- コピーせずに変更すると呼び出し側に影響する
- キャプチャの追加の影響はマッチに成功した場合だけに限る
- /(?<n>a)b|ac/ =~ "ac"
- try(
  [:alt, [:cat, [:capture, :k, "a"], "b"],
  [:cat, "a", "c"]],
  %w[a c], 0, {}) {|pos, md| p [pos, md] }
  #=> [2, {}]
- /(?<n>a)b|ac/ で、ac の部分にマッチするキャプチャの所は通らない

コピーの必要性

コピーせずに共有してると
外に影響が出る

/(?<n>a)b|ac/ =~ "ac"

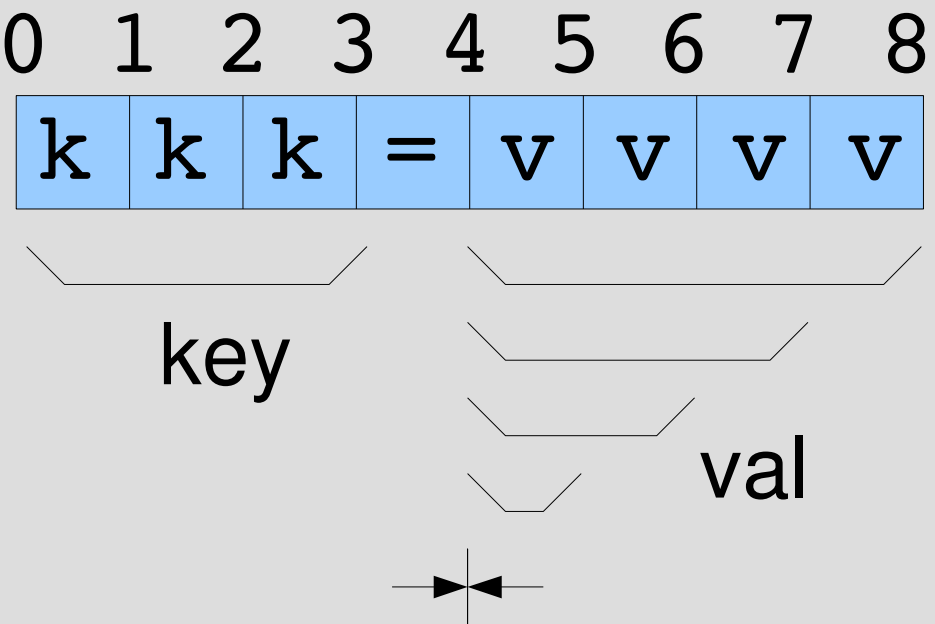


try_capture の実行 /¥A(?<n>a*)/ =~ "aab"

- try([:capture, :n, [:rep, "a"]],
  %w[a a b], 0, {}) {|pos, md|
  p [pos, md]
  }
  #=>
  [2, {:n=>0...2}]
  [1, {:n=>0...1}]
  [0, {:n=>0...0}]

try_capture の実行 (2)
/¥A(?<key>k*)=(?<val>v*)/

- try([:cat, [:capture, :key, [:rep, "k"]],
  [:cat, "=", [:capture, :val, [:rep, "v"]]]],
  %w[k k k = v v v v], 0, {}) {|pos, md|
  p [pos, md]
  }
  #=>
  [8, {:val=>4...8, :key=>0...3}]
  [7, {:val=>4...7, :key=>0...3}]
  [6, {:val=>4...6, :key=>0...3}]
  [5, {:val=>4...5, :key=>0...3}]
  [4, {:val=>4...4, :key=>0...3}]



任意の一文字

- /¥A(?<key>k*)=(?<val>v*)/ は非実用的
- key には k しか使えない
- val には v しか使えない
- とりあえず任意の文字列にしたい
- /¥A(?<key>.*)=(?<val>.*)/
- . を try でサポートする

[:anychar] の基本動作

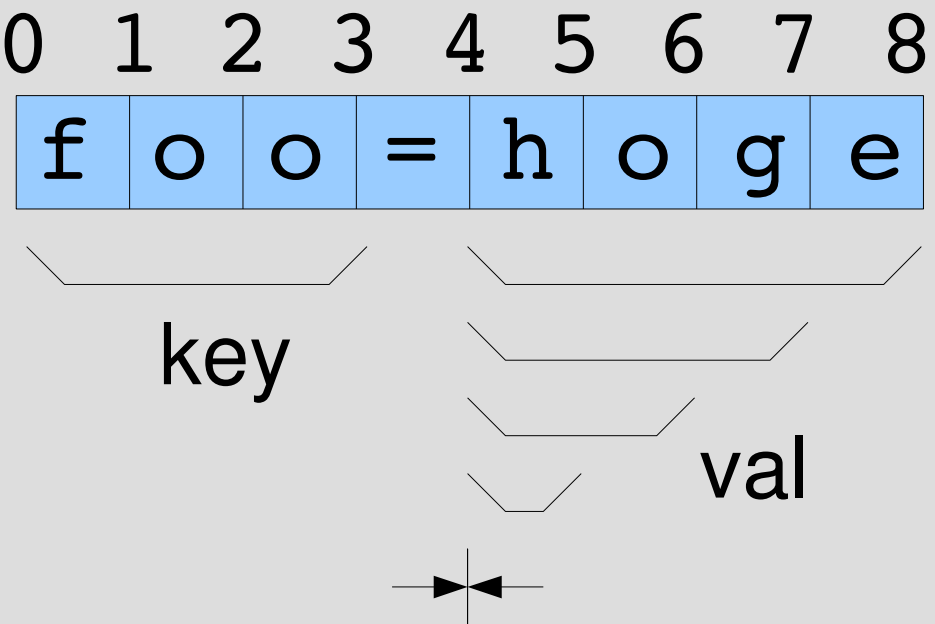
- try([:anychar], %w[b a n a n a],0,{}){|pos,md|
  p pos
  }
  #=> 1

[:anychar] の使用例 /¥A(?<key>k*)=(?<val>v*)/

- try([:cat, [:capture, :key, [:rep, [:anychar]]],
  [:cat, "=",
  [:capture, :val, [:rep, [:anychar]]]]],
  %w[f o o = h o g e], 0, {}) {|pos, md|
  p [pos, md]
  }

#=>

[8, {:val=>4...8, :key=>0...3}]
[7, {:val=>4...7, :key=>0...3}]
[6, {:val=>4...6, :key=>0...3}]
[5, {:val=>4...5, :key=>0...3}]
[4, {:val=>4...4, :key=>0...3}]



[:anychar] の実装

- when :anychar
  yield pos+1, md if pos < str.length
- 文字列の終端より前なら pos+1, md を yield

レポート

- キャプチャに対応したtryで連接と選択を任意個引数に拡張せよ

- 実装したらユニットテストで確認せよ
- 〆切 2008-06-17 12:00
- RENANDI
- 拡張子が txt なテキストファイルがよい

任意個引数の連接の例

```
try([:cat,
     [:capture, :key, [:rep, [:anychar]]],
     "=",
     [:capture, :val, [:rep, [:anychar]]]],
    %w[f o o = h o g e], 0, {}) {|e, md|
  p [e, md]
}
```

出力:

```
[8, {:key=>0...3, :val=>4...8}]
[7, {:key=>0...3, :val=>4...7}]
[6, {:key=>0...3, :val=>4...6}]
[5, {:key=>0...3, :val=>4...5}]
[4, {:key=>0...3, :val=>4...4}]
```

任意個引数の選択の例

```
try([:alt,
      [:capture, :foo, [:rep, "a"]],
      [:capture, :bar, [:rep, "b"]],
      [:capture, :baz, [:rep, "c"]]],
    %w[b b b], 0, {}) {|pos, md| p [pos, md] }
#=>
[0, {:foo=>0...0}]
[3, {:bar=>0...3}]
[2, {:bar=>0...2}]
[1, {:bar=>0...1}]
[0, {:bar=>0...0}]
[0, {:baz=>0...0}]
```

まとめ

- 前回のレポートの説明
- キャプチャの説明
- MatchData
- キャプチャの実装
- 任意の一文字
- レポート