BioRuby その1

Rubyによる生物情報解析の基礎

2003年1月28日

BioRubyプロジェクト
大阪大学 遺伝情報実験センター

後藤　直久

ng@bioruby.org
ngoto@gen-info.osaka-u.ac.jp

Rubyとは？(1)

- オブジェクト指向スクリプト言語
  - 1993年2月誕生, 1995年12月公開
  - スクリプト言語の柔軟性
    - Perlで出来ることはひととおり出来ることを目指す
  - 純粋なオブジェクト指向言語
    - プログラムは書きやすく読みやすい
- まつもとゆきひろ氏が開発
  - 本人曰く「15年来の言語マニア」
  - PerlやPythonなど既存の言語に不満
  - 各言語の「いいとこどり」？

Rubyとは？(2)

- 日本発最初(?)のメジャープログラミング言語
  - 日本で出来たので日本語のドキュメントが多い
    - 最近は逆転されつつあるかも
- 日本だけでなく全世界へ広まる
  - Ruby Conference
    - アメリカで2001年から毎年開催
    - 参加者はほとんどアメリカ人（日本からは数名）
  - メーリングリストの投稿数は日本語<<英語
    - Ruby Garden などネット上の活動も盛ん

(私にとって)なぜRubyか？

(1999年頃の話)

- 昔(1995年)見て気にはなっていた
- Rubyのほうがわかりやすそうだった
- オブジェクト指向を知ってみたかった
- 日本語のドキュメントがウェブ上に多かった
- Perlは一応知っていたがいまいち苦手だった
- Rubyなら大規模なプログラムも書きやすそう
- Perlの代替？

Rubyでやる意義

- Rubyはすべてがオブジェクト
  - データ構造を自然に表現
  - 生物学はデータの塊
- スクリプトを書きやすく読みやすい
  - 開発効率が高い
  - 情報科学に詳しくない人にもわかりやすい
- 拡張モジュールを(C言語で)書きやすい
  - パワーが必要な処理は拡張モジュールへ
  - 解析のプラットフォームとしての利用

BioRubyとは？

- バイオインフォマティクスに必要な機能や環境をRubyを用いて統合的に実装することを目指したプロジェクト
- Rubyによるバイオインフォマティクスに必要な部品を揃えたライブラリ
- 2000年11月開始

http://bioruby.org/

BioRubyの機能(1)

- 塩基・アミノ酸配列の操作
  - 翻訳、スプライシング、検索など
- データベースエントリの読み込み
  - GenBank, DDBJ, EMBL, SwissProt, KEGG, Prosite, TRANSFAC, Aaindex, ...
- 各種解析ソフトウェアによる解析の支援
  - BLAST, FASTA, HMMER
    - 自作スクリプト内部からの呼び出し
    - 結果の解析支援

BioRubyの機能(2)

- ファイルやネットからのデータ読み込み
  - BioFetch
  - BioSQL
  - BioFlat
- 二項関係(パスウェイ)
  - Bio::Pathway, Relation
- 文献データなど
  - MEDLINE

Rubyのインストール

- 注意: BioRubyにはRuby 1.6.6以降が必要です
  （現在の最新安定版は1.6.8）
- Linux, FreeBSD, SolarisなどUNIX系
  - 多くの場合はパッケージに含まれています
  - 無い場合やバージョンが古い場合は自分でmake
- Windows系
  - http://www.ruby-lang.org/ からバイナリをダウンロード
  - 3種類あるが、BioRubyではCygwin版がよさそう
- MacOS X
  - 最初から入っています(10.2以降？)
  - 自分でmakeしても, パッケージを使用してもOK

BioRubyのインストール

1. http://bioruby.org/ からダウンロード
   （最新リリース版は 0.4.0）
2. コマンドラインから
   % ruby install.rb config
   % ruby install.rb setup

ここでスーパーユーザ権限になって
   # ruby install.rb install

他のライブラリへの依存は一応ないが、できれば
XMLparserまたはREXMLのインストールを推奨。

まず、簡単な例から

sample01.rb

```
#!/usr/bin/env ruby
# DNAを変数に格納し、表示する

# まず、DNAを変数dnaに格納する
dna = 'ACGGGAGGACGGGAAAATTAACTACGGCATTAGC'

# 次にDNA(変数dna)を表示する
print dna

# 最後にプログラムを明示的に終了させる
exit
```

同じプログラムをPerlで書く

```
#!/usr/bin/perl -w
# DNAを変数に格納し、表示する

# まず、DNAを変数$DNAに格納する
$DNA = 'ACGGGAGGACGGGAAAATTAACTACGGCATTAGC';

# 次にDNA(変数$DNA)を表示する
print $DNA;

# 最後にプログラムを明示的に終了させる
exit;
```

(『バイオインフォマティクスのためのPerl入門』p.36 例題4-1, コメントを一部改変)

sample02.rb　　Rubyプログラムの例

```
#!/usr/bin/env ruby
# DNAをつなげる

# dna1, dna2 という2つの変数に2つのDNA断片を格納
dna1 = 'ATGCGAGGTTGTTGAAGTCGATGTCCTACCAGGAAGCG'
dna2 = 'AAGCTGCCGTCAGAACC'

# 元のDNA断片を表示
print "DNA1: ", dna1, "¥n"
print "DNA2: ", dna2, "¥n¥n"

# 文字列連結
# +(プラス)演算子
dna3 = dna1 + dna2
print "DNA3: ", dna3, "¥n"

# 「式展開」
dna4 = "#{dna1}#{dna2}"
print "DNA4: ", dna4, "¥n"

if dna3 == dna4 then
    print "DNA3 と DNA4 は同じ。¥n"
else
    print "DNA3 と DNA4 は違う。¥n"
end
```

同じ内容をPerlで書く

```
#!/usr/bin/perl -w
# DNAをつなげる

# $dna1, $dna2 という2つの変数に2つのDNA断片を格納
$dna1 = 'ATGCGAGGTTGTTGAAGTCGATGTCCTACCAGGAAGCG';
$dna2 = 'AAGCTGCCGTCAGAACC';

# 元のDNA断片を表示
print "DNA1: ", $dna1, "¥n";
print "DNA2: ", $dna2, "¥n¥n";

# 文字列連結
# .(ドット)演算子
$dna3 = $dna1 . $dna2;
print "DNA3: ", $dna3, "¥n";

# 「文字列展開」
$dna4 = "$dna1$dna2";
print "DNA4: ", $dna4, "¥n";

if ($dna3 eq $dna4) {
    print "DNA3 と DNA4 は同じ。¥n";
} else {
    print "DNA3 と DNA4 は違う。¥n";
}
```

informatics BioRuby.org

RubyのPerlとの主な違い

- 行末のセミコロン(;)は不要(有ってもよい)
- 変数は先頭がアルファベット小文字
  - 何も付けないとローカル変数
  - 先頭に$を付けるとグローバル変数
  - 先頭がアルファベット大文字なのは定数
- 文字列連結は + で、比較は == である
- 式展開は”#{a}”のように #{ } で変数等を囲う
  - 「式」なので”#{a[3]}”や”#{a.tr('atgc', 'tacg')}”が可能
- if ブロックの終わりとして end を使う

RubyのPerlとの主な違い

- ローカル変数のスコープはブロック内
  - Perlでは必須の my はRubyには存在しない
- ローカル変数を代入する前に使用するとエラーに
  - いきなり print a → エラー
  - a = 1; print a → OK
- 整数(おおよそ-2^30～2^30の範囲)以外はいわゆる参照渡しに
- スカラーコンテキストとかリストコンテキストとかは気にしなくてよい
  - 配列が必要な所では明示的に配列を使う

考え方の違い

- Perl
  - 演算や関数などの操作を変数に適用
    - 手続き型
- Ruby
  - オブジェクトにメッセージを送る
    - オブジェクト指向

```
$s = 'AAGTCGTAACAAGGTA';
$s =~ tr/ACGT/TGCA/;
reverse $s;
print $s, "¥n";
print length($s), "¥n";
```

```
s = 'AAGTCGTAACAAGGTA'
s.tr!('ACGT', 'TGCA')
s.reverse!
print s, "¥n"
print s.length, "¥n"
```

用語説明

- オブジェクト (Object)
  - Rubyの扱うすべてのデータ
  - メッセージを理解
    - **例: str.length**
      - **str** というオブジェクトに **length** というメッセージを渡す
- メソッド (Method)
  - メッセージに対応する処理のこと
    - メッセージを受け取るオブジェクトを「レシーバ」と言う
    - 例: **str.length** の **str** はレシーバ, **length** はメソッド

用語説明

- クラス (Class)
  - オブジェクトの種類を表す
    - 例: 文字列 "I am a boy." はStringクラスのオブジェクト
- インスタンス (Instance)
  - あるクラスに属するオブジェクトのこと
    - クラスに属することを強調する場合に使用
- モジュール (Module)
  - メソッドや定数をまとめたもの

メソッドの種類

- 通常のメソッド
  - オブジェクトのコピーを作成し、そこに結果を反映
  - 元のオブジェクトには触らずそのまま
    - a="agtcgtagat"; b=a.gsub(/ag/, 'ct')
- 破壊的メソッド
  - そのオブジェクト自体を変更
  - 副作用はあるが効率は良い
  - メソッド名の最後に!が付いていることが多い
    - a="agtcgtagat"; a.gsub!(/ag/, 'ct')

変数の代入

- 整数とtrue,nil,falseは値そのものを代入
- それ以外はいわゆる参照渡しである

sample05.rb

```
a = "atgcatgc"
b = a
print b, "¥n"

a << "aag"
a.gsub!(/t/, "u")

print b, "¥n"
```

aを変更したはずなのに、
bも変更されている

sample06.rb

```
a = "atgcatgc"
b = a.dup
print b, "¥n"

a << "aag"
a.gsub!(/t/, "u")

print b, "¥n"
```

別物として扱いたいなら
明示的にコピー(dup)しよう

Bio::Sequence::NA クラス

塩基配列を格納するクラス

Bio::Sequence::NA.new(文字列)でオブジェクトを新規作成

作成後は文字列と同様に扱える(Stringのメソッドを利用可能)

sample10.rb

```
#!/usr/bin/env ruby

require 'bio'

s1 = Bio::Sequence::NA.new('ATGGCATTAGTTTGTTG')
print s1.to_fasta("Sample1", 70)

s2 = s1 + "ATGC" * 10
print s2.to_fasta("Sample2", 70)
```

Bio::Sequence::NA クラス

主なメソッド一覧

to_fasta(label, width)
　　FASTAフォーマットに変換。widthは省略時無限大。

subseq(from, to)
　　部分配列を得る。

spliceing(position)
　　スプライシングを行う。"1..100"や"complement(join(1..10,20..30))"
　　のような形式で指定

composition
　　組成をハッシュとして返す。

complement
　　相補鎖を返す。

translate(frame = 1, table = 1)
　　アミノ酸への翻訳を行う。frame, tableは省略可能。
　　後述のBio::Sequence::AAクラスのインスタンスを作成.。

Bio::Sequence::AA クラス

アミノ酸配列を格納するクラス

Bio::Sequence::AA.new(文字列)でオブジェクトを新規作成

作成後は文字列と同様に扱える(Stringのメソッドを利用可能)

sample11.rb

```
#!/usr/bin/env ruby

require 'bio'

a1 = Bio::Sequence::AA.new('MGNTKLANPAPLGLMGFGMTTI')
print a1.to_fasta("SampleAA1", 70)

a2 = a1 + "QMNAPGSTKV"
print a2.to_fasta("SampleAA2", 70)
```

Bio::Sequence::AA クラス

主なメソッド一覧

to_fasta(label, width)
　　FASTAフォーマットに変換。widthは省略時無限大。

subseq(from, to)
　　部分配列を得る。

composition
　　組成をハッシュとして返す。

codes
　　3文字表記を返す

molecular_weight
　　分子量を返す

Bio::FlatFileクラス

フラットファイルからデータを読み込むクラス

例:あらゆる入力ファイルをFASTA形式に変換して表示

使い方: % ruby to_fasta.rb filename...

```
#!/usr/bin/env ruby
require 'bio'

ff = Bio::FlatFile.new(nil, ARGF)

ff.each do |x|
   print x.seq.to_fasta(x.entry_id, 70)
end
```

Bio::FlatFileクラス

Bio::FlatFile.open(ファイル形式, ファイル名)　または
Bio::FlatFile.new(ファイル形式, ファイルオブジェクト) で作成。

ファイル形式はBioRubyのデータベースのクラスで指定。

主なクラス一覧

GenBank	GenBank形式のクラス。
DDBJ	DDBJ。GenBank形式と同じ。
EMBL	EMBL形式のクラス。
GenPept	GenPept形式。
SPTR	SwissProtとTrEMBLの形式

ファイル形式に nil を指定すると自動判別する！

Bio::FlatFileの応用
ファイル形式の判別

使い方: % ruby biofile.rb filename...

```
#!/usr/bin/env ruby
require 'bio'

ARGV.each do |arg|
    ff = Bio::FlatFile.open(nil, arg)
    # 自動判別に失敗していた場合は再度試みる
    ff.autodetect unless ff.dbclass
    # 認識されたクラスを得る
    klass = ff.dbclass
    if klass then
        print arg, ": ", klass.to_s.sub(/^Bio::/, ''), "¥n"
    else
        print arg, ": unknown¥n"
    end
    ff.close
end
```

イテレータ：eachによる繰り返し

- イテレータは繰り返しを抽象化したもの
  - ファイルなら1行ずつ
  - 配列(Array)なら1要素ずつ

```
a = File.open("filename")
a.each do |x|
    print x
end
a.close
```

```
a = [ 3, 1, 4, 1, 5, 9, 2 ]
a.each do |x|
    print x * 2, "¥n"
end
```

- while や for などと比較して
  - iなどの繰り返しを制御する変数が不要
  - 繰り返しの開始・終了条件などの設定が不要

繰り返しの「中身」に集中できる
違うデータ構造でも同一ルーチンで処理可能

Bio::FlatFileの応用例
簡易grep

使い方: % ruby biogrep.rb 正規表現 filename...

```
#!/usr/bin/env ruby
require 'bio'

re = Regexp.new(ARGV.shift, Regexp::IGNORECASE)

ARGV.each do |arg|
    ff = Bio::FlatFile.open(nil, arg)
    ff.each do |x|
        if p = (re =~ x.seq) then
            print x.entry_id, " : ", p+1, "¥n"
        end
    end
    ff.close
end
```

ただし、実行スピードは遅いです。

Bio::FlatFileの応用例
A,C,G,Tの含有率を表示

使い方: % ruby contents.rb filename...

```
#!/usr/bin/env ruby
require 'bio'

ARGV.each do |arg|
    ff = Bio::FlatFile.open(nil, arg)
    ff.each do |x|
        # Accession 長さ A C G T の順にタブ区切り出力
        c = x.naseq.composition
        len = x.naseq.length
        print [
                x.entry_id,
                len,
                c['a'] * 100.0 / len,
                c['c'] * 100.0 / len,
                c['g'] * 100.0 / len,
                c['t'] * 100.0 / len,
            ].join("¥t"), "¥n"
    end
    ff.close
end
```

データベースエントリ

GenBank形式

Bio::GenBank
クラス

```
LOCUS       AAB2MCG1                289 bp    DNA     linear   PRI 23-AUG-2002
DEFINITION  Aotus azarai beta-2-microglobulin precursor exon 1.
ACCESSION   AF032092
VERSION     AF032092.1  GI:3265027
KEYWORDS    .
SEGMENT     1 of 2
SOURCE      Aotus azarai (Azara's night monkey)
  ORGANISM  Aotus azarai
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;
            Mammalia; Eutheria; Primates; Platyrrhini; Cebidae; Aotinae; Aotus.
REFERENCE   1  (bases 1 to 289)
  AUTHORS   Canavez,F.C., Ladasky,J.J., Muniz,J.A., Seuanez,H.N., Parham,P. and
            Cavanez,C.
  TITLE     beta2-Microglobulin in neotropical primates (Platyrrhini)
  JOURNAL   Immunogenetics 48 (2), 133-140 (1998)
  MEDLINE   98298008
   PUBMED   9634477
REFERENCE   2  (bases 1 to 289)
  AUTHORS   Canavez,F.C., Ladasky,J.J., Seuanez,H.N. and Parham,P.
  TITLE     Direct Submission
  JOURNAL   Submitted (31-OCT-1997) Structural Biology, Stanford University,
 Fairchild Building Campus West Dr. Room D-100, Stanford, CA
            94305-5126, USA
FEATURES             Location/Qualifiers
     source          1..289
                     /organism="Aotus azarai"
                     /db_xref="taxon:30591"
     sig_peptide     134..193
     exon            <134..200
                     /number=1
     intron          201..>289
                     /number=1
BASE COUNT       30 a     99 c     80 g     80 t
ORIGIN
        1 gtccccgcgg gccttgtcct gattggctgt ccctgcgggc cttgtcctga ttggctgtgc
       61 ccgactccgt ataacataaa tagaggcgtc gagtcgcgcg ggcattactg cagcggacta
      121 cacttgggtc gagatggctc gcttcgtggt ggtggccctg ctcgtgctac tctctctgtc
      181 tggcctggag gctatccagc gtaagtctct cctcccgtcc ggcgctggtc cttcccctcc
      241 cgctcccacc ctctgtagcc gtctctgtgc tctctggttt cgttacctc
//
```

データベースエントリ
例: Bio::GenBankクラス

主なメソッド

entry_id
　　データ固有のID
naseq
　　塩基配列(無いときはエラーに) Bio::Sequence::NAオブジェクト
aaseq
　　アミノ酸配列(無いときはエラーに) Bio::Sequence::AAオブジェクト
locus
　　LOCUS行の内容
accession
　　アクセッション番号
version
　　バージョン番号
gi
　　GI番号
features
　　アノテーション情報など

Bio::FeaturesとBio::Locations

例：FEATURE内の相対位置も表示する簡易grep

使い方: % ruby biogrep2.rb 正規表現 filename...

```
#!/usr/bin/env ruby
require 'bio'

re = Regexp.new(ARGV.shift, Regexp::IGNORECASE)

ARGV.each do |arg|
    ff = Bio::FlatFile.open(nil, arg)
    ff.each do |x|
        if p = (re =~ x.seq) then
            print x.entry_id, " : ", p+1, "¥n"
            x.features.each do |y|
                if relpos = y.locations.relative(p+1) then
                    # 種類, 位置, 相対位置 の順にタブ区切りで表示
                    print "¥t", [ y.feature, y.position,
                        relpos ].join("¥t"), "¥n"
                end
            end
        end
    end
    ff.close
end
```

BioFlat

- フラットファイルのインデックスシステム
  - フラットファイルにインデックスを付けることで、高速にエントリの検索と取得が可能となる
  - OBDA準拠のファイル形式なので互換性も安心
  - bioflatコマンドでインデックス作成とエントリの検索・取得が可能

インデックス作成の例

```
% bioflat -makeindex INDEX --files /db/gbhtg*.seq
```

エントリ取得の例

```
% bioflat  INDEX ABC12345
```

BioFlat応用例

- 自分独自の2次データベース構築
- ウェブ経由で全世界に公開も