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

BioRuby

片山 俊明，五斗 進
京大化研バイオインフォマティクスセンター

BioRuby プロジェクトとは

- 国産の優れたオブジェクト指向スクリプト言語 Ruby を使って、Bioinfomatics の面倒なルーチンワークを再利用できる形でライブラリ化する
  - 生物学者がいつもの仕事をサラッと片付けられる
    - データベースからエントリを取ってきてパースする
    - Blast をガンガン流して結果を回収する
- 海外の先行プロジェクト BioPerl, BioJava, BIoPython 等と比較しての利点
  - Ruby という言語が高機能で使いやすい、短く書ける
  - 開発コミュニティが日本語を話す

Open Bio*

- O|B|F -- Open Bio Foundation
  - バイオインフォマティクスに関するオープンソースコミュニティ

- BioRuby
- BioPerl
- BioPython
- BioJava
- BioDAS
- BioMOBY
- EMBOSS

- Ensembl
- OmniGene
- GMOD
- Apollo
- OBDA

- BioCaml
- BioLisp
- BioConductor
- BioPathways
- BioBlog
- BioCyc
- BioDog

:

OBDA

- BioHackathon
  - 2002/01 Arizona, 2002/02 Cape Town
  - どの Open Bio* ライブラリで作成したデータベースも共通にアクセスできる
  - 競合ではなく協調 Open Bio* ハッカーは仲間です:-)

OBDA

- BioHackathon
  - 2002/01 Arizona, 2002/02 Cape Town
  - どの Open Bio* ライブラリで作成したデータベースも共通にアクセスできる
  -

OBDA

- BioHackathon
  - 2002/01 Arizona, 2002/02 Cape Town
  - どの Open Bio* ライブラリで作成したデータベースも共通にアクセスできる
  - 競合ではなく協調 Open Bio* ハッカーは仲間です:-)
- Open Bio* Sequence Database Access
  - Directory Registry (Stanza)
  - Flat File indexing (DBM, BDB)
  - BioFetch (CGI/HTTP)
  - BioSQL (MySQL, PostgreSQL, Oracle)
  - SOAP (XEMBL based)
  - BioCORBA (BSANE compliant)

OBDA を使ったエントリ取得

- 設定ファイル (Stanza フォーマット)
  - ~/.bioinformatics/seqdatabase.ini
  - /etc/bioinformatics/seqdatabase.ini
  - http://open-bio.org/registry/seqdatabase.ini

```
[swissprot]
protocol=biosql
location=db.bioruby.org
dbname=biosql
driver=mysql
biodbname=sp

[embl]
protocol=biofetch
location=http://bioruby.org/cgi-bin/biofetch.rb
biodbname=embl
  :
```

OBDA を使ったエントリ取得

- 設定ファイル (Stanza フォーマット)
  - ~/.bioinformatics/seqdatabase.ini
  - /etc/bioinformatics/seqdatabase.ini
  - http://open-bio.org/registry/seqdatabase.ini

```
[swissprot]
protocol=biosql
location=db.bioruby.org
dbname=biosql
driver=mysql
biodbname=sp

[embl]
protocol=biofetch
location=http://bioruby.org/cgi-bin/biofetch.rb
biodbname=embl
  :
```

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

reg = Bio::Registry.new
db = reg.db("swissprot")
entry = db.fetch("TETW_BUTFI")
```

BioRuby にできること

- Bio::Sequence, Bio::Location, Bio::Feature クラス
  - 塩基配列、アミノ酸配列の取り扱い
  - 組成、スプライシング、翻訳、ウィンドウサーチなど
- Bio::DB クラス
  - データベースのパーザ（現在２０種くらい）
- Bio::Blast, Bio::Fasta クラス
  - Blast/Fasta を実行し結果をパーズするファクトリ
- Bio::PubMed, Bio::Reference クラス
  - 文献データベース検索、BibTeX などのフォーマッタ
- Bio::Registry, Bio::SQL, Bio::Fetch, Bio::FlatFile クラス
  - データベース入出力インターフェイス（OBDA 準拠）
- Bio::Pathway, Bio::Relation クラス
  - グラフ、２項関係の演算

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate    ➔ "HELL*XW*RLD"

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

```
         ➔ ["HELL*", "W*RLD"]
```

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

➔ "HELL* W*RLD"

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

```
➔ "Hell* w*rld"
```

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

➔ "Hello world"

informatics BioRuby.org

初めて実行するプログラム

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

require 'bio'

gene = Bio::Seq::NA.new("catgaattattgtagannntgataaagacttgac")
prot = gene.translate

puts plot.split('X').join(' ').capitalize.gsub(/¥*/, 'o') << '!'
```

➔ Hello world!

コード比較

- FASTA ファイルを BioRuby で読み込む

```
#!/usr/bin/ruby

require 'bio'

flatfile = Bio::FlatFile.open(Bio::FastaFormat, 'filename')

flatfile.each do |entry|
  puts entry.entry_id
  puts entry.seq
  puts entry
end
```

・ FASTA ファイルを BioPerl で読み込む

```
#!/usr/bin/perl

use Bio::SeqIO;

my $seqio = new Bio::SeqIO(-format => 'fasta',
                           -file => 'filename');

while ( my $entry = $seqio->next_seq ) {
  print $entry->display_id, "¥n";
  print $entry->seq, "¥n";
  print ">", $entry->desc, "¥n", $entry->seq, "¥n";
}
```

・ FASTA ファイルを BioPython で読み込む

```
#!/usr/bin/python

from Bio import Fasta

iter = Fasta.Iterator(open('filename'), Fasta.RecordParser())

while 1:
    entry = iter.next()
    if not(entry): break
    print entry.title
    print entry.sequence
    print entry
```

コード比較

- BioRuby で Blast を local に実行

```
#!/usr/bin/ruby

require 'bio'

blast = Bio::Blast.local('blastp', 'hoge.pep')
flatfile = Bio::FlatFile.open(Bio::FastaFormat, 'queryfile')

flatfile.each do |seq|
  result = blast.query(seq)
  result.each do |hit|
    puts hit.query_id, hit.target_id, hit.evalue if hit.evalue < 0.05
  end
end
```

- BioPerl で Blast を local に実行

```
#!/usr/bin/perl

use Bio::SeqIO;
use Bio::Tools::Run::StandAloneBlast;
use Bio::Tools::BPlite;

my @params = ('program' => 'blastp', 'database' => 'hoge.pep');
my $factory = Bio::Tools::Run::StandAloneBlast->new(@params);

my $input = Bio::SeqIO->new(-format => 'fasta', -file => "queryfile");

while ( my $seq = $input->next_seq ) {
  $result = $factory->blastall($seq);
  while ( my $hit = $result->nextSbjct ) {
    while ( my $hsp = $hit->nextHSP ) {
      print $result->query, $hit->name, $hsp->P, "¥n" if $hsp->P < 0.05;
      last;
    }
  }
}
```

- BioPython で Blast を local に実行

```
#!/usr/bin/python

from Bio import Fasta
from Bio.Blast import NCBIStandalone

iterator = Fasta.Iterator(open("queryfile"), Fasta.RecordParser())
while 1:
    query = iterator.next()
    if not(query): break
    open("query.fst", "w").write(str(query))
    out, error = NCBIStandalone.blastall("blastall", "blastp", "hoge.pep", "query.fst")
    parser = NCBIStandalone.BlastParser()
    result = parser.parse(out)
    for alignment in result.alignment:
        for hsp in alignment.hsps:
            if hsp.expect < 0.05:
                print query.title, alignment.title, hsp.expect
```

いまからの課題

- ドキュメント
- SOAP(DAS, XEMBL, 遺伝研), CORBA アクセスの整備
- HMMER, EMBOSS, ClustalW, T-Coffee など対応
- PDB – 構造系のデータベースクラス、メソッドの開発
- PATHWAY, SSDB, KO, GO, InterPro
- BioFetch で Entrez E-utils 対応
- GFF, AGAVE, GAME フォーマット
- 配列データベース系クラスのリファクタリング
- アライメントクラス

BioRuby.org

- 総合情報 http://bioruby.org/
- 開発情報 http://ura.bioruby.org/
- ニュース http://q--p.bioruby.org/

- ソース　 ftp://bioruby.org/
- ＣＶＳ　 cvs.bioruby.org

- ＭＬ　　 ja@bioruby.org, dev@bioruby.org
- 問い合せ staff@bioruby.org

  – presentation by T. Katayama <k@bioruby.org>