KEGG API

KEGG API はプログラムなどから KEGG を利用するためのウェブサービスです。 前半では、KEGG データベースから情報を取得したり検索したりするために KEGG API を使う方法を説明します。後半のリファレンスで KEGG API の全機能を 解説します。例として主に Ruby 言語を使って解説しますが、SOAP と WSDL を 扱うことのできる言語（Perl, Python, Java など）であれば簡単に KEGG API を 利用することができます。

目次

イントロダクション

ウェブサービスとは、クライアントからの要求をインターネットを介してサーバに 送り、サーバがプログラムの実行結果をクライアントに返す仕組みで、一般的には ウェブページで使われる HTTP プロトコルと、構造を持つデータの表現方法として 普及している XML マークアップ文書形式を用いたものを指します。

ウェブサービスはプログラムから利用できるため、定期的に検索を行ったり、 少しずつ値を変えた様々な要求を自動的に処理したりするのに向いています。 このため、株価や天気情報の問い合わせ、Google への複合検索などでも使われて います。

HTTP を用いるメリットには、誰でも使えることやファイアウォールなどの制限を 受けにくいことがあり、XML の方には関連技術が揃っていることや複雑なデータ 構造を表現できるといったポイントがあります。

ウェブサービスでは XML 関連技術の中でも SOAP と WSDL を使うことが多くなって います。SOAP はクライアントとサーバがやりとりするメッセージの表現方法を 標準化したもので、以前は Simple Object Access Method の略とされていました (今は Service Oriented Access Protocol ということもあるようです)。 WSDL は SOAP に基づくサービスをコンピュータが簡単に利用できるようにする ためのもので、Web Service Description Language の略となっています。

KEGG API はこれらの技術を使って、自分の興味ある遺伝子やパスウェイなどの 情報を自由に検索したり解析に用いたりするための手段を提供します。ユーザは KEGG の多くの機能を、ウェブページをクリックする代わりに自分のプログラムの 中から次々と実行することができるようになります。

KEGG API に関する最新の情報は以下の URL から得ることができます。

- <URL:http://www.genome.jp/kegg/soap/>

KEGG API の使い方

以下では Ruby, Perl, Python, Java の各言語による KEGG API の簡単な使い方を 紹介します。各言語で SOAP と WSDL を扱えるライブラリを追加インストールする 必要があります。

Ruby の場合

Ruby 1.8.1 以降では、標準で SOAP を使う事ができますので追加インストー ルは必要ありません。

Ruby 1.8.0 では SOAP4R, devel-logger, http-access2 などのライブラリをインストールする必要があります。

Ruby 1.6.8 の場合はさらに SOAP4R が必要とする他のライブラリ (date2, uconv, XML のパーザなど) もインストールする必要がありますので、あらかじめ SOAP4R のドキュメントに従って入れておきます。

以下のサンプルコードは、大腸菌の b0002 遺伝子と最も相同性の高い遺伝子 を、Smith-Waterman スコアの高い順に 5 個検索して表示するプログラムです。

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

require 'soap/wsdlDriver'

wsdl = "http://soap.genome.jp/KEGG.wsdl"
serv = SOAP::WSDLDriverFactory.new(wsdl).create_driver
serv.generate_explicit_type = true      # SOAP と Ruby の型変換を有効にする

start = 1
max_results = 5

top5 = serv.get_best_neighbors_by_gene('eco:b0002', start, max_results)
top5.each do |hit|
  print hit.genes_id1, "¥t", hit.genes_id2, "¥t", hit.sw_score, "¥n"
end
```

プログラムの中で 'get_best_neighbors_by_gene' は、KEGG の SSDB データ ベースを使って KEGG の GENES に含まれている各生物種の中から最も相同性 の高い遺伝子を探してくる API です。結果は次のように表示されます。

```
eco:b0002       eco:b0002       5283
eco:b0002       ecj:JW0001      5283
eco:b0002       sfx:S0002       5271
eco:b0002       sfl:SF0002      5271
eco:b0002       ecc:c0003       5269
```

うまく動かない場合は、

```
serv = SOAP::WSDLDriverFactory.new(wsdl).create_driver
serv.wiredump_dev = STDERR      # ←この行を書き足す
serv.generate_explicit_type = true
```

のように wiredump_dev に STDERR を指定した行を追加して実行することで、 サーバとのやり取りが標準エラーに出力されます。

KEGG API v3.0 から、サーバの負担を軽くしたりタイムアウトを防ぐ目的で、 大量の結果を返すメソッドには start, max_results 引数が導入され、一度に 得られる結果の数が制限されるようになりました。このため、これらのメソッ ドで全ての結果を得るためにはループを用いる必要があります。

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

require 'soap/wsdlDriver'

wsdl = "http://soap.genome.jp/KEGG.wsdl"
serv = SOAP::WSDLDriverFactory.new(wsdl).create_driver
serv.generate_explicit_type = true

start = 1
max_results = 100

loop do
  results = serv.get_best_neighbors_by_gene('eco:b0002', start, max_results)
  break unless results  # 結果が返ってこなければ終了
  results.each do |hit|
    print hit.genes_id1, "¥t", hit.genes_id2, "¥t", hit.sw_score, "¥n"
  end
  start += max_results
end
```

WSDL を用いているため、これらの例でも Ruby の場合は十分に簡単に書けま すが、BioRuby を使うとさらにスッキリ書く ことが

できます。

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

require 'bio'

serv = Bio::KEGG::API.new

results = serv.get_all_best_neighbors_by_gene('eco:b0002')
results.each do |hit|
  print hit.genes_id1, "¥t", hit.genes_id2, "¥t", hit.sw_score, "¥n"
end
```

BioRuby では 'get_all_best_neighbors_by_gene' メソッドが定義されており、 自動で上記の例のループを回して全ての結果を返してくれます。また、取り出 したい名前のリストを渡すと対応する値を配列で返してくれる filter メソッ ドを使うこともできます。

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

require 'bio'

serv = Bio::KEGG::API.new

results = serv.get_all_best_neighbors_by_gene('eco:b0002')

# 欲しい値が遺伝子名のペアと SW スコアだけの場合の例
fields = [:genes_id1, :genes_id2, :sw_score]
results.each do |hit|
  puts hit.filter(fields).join("¥t")
end

# それぞれの遺伝子でアライメントされたポジションなども表示させる例
fields1 = [:genes_id1, :start_position1, :end_position1, :best_flag_1to2]
fields2 = [:genes_id2, :start_position2, :end_position2, :best_flag_2to1]
results.each do |hit|
  print "> score: ", hit.sw_score, ", identity: ", hit.identity, "¥n"
  print "1:¥t", hit.filter(fields1).join("¥t"), "¥n"
  print "2:¥t", hit.filter(fields2).join("¥t"), "¥n"
end
```

次は、大腸菌 (eco) に対する KEGG パスウェイの一覧を返す例です。

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

require 'bio'

serv = Bio::KEGG::API.new

list = serv.list_pathways("eco")
list.each do |path|
  print path.entry_id, "¥t", path.definition, "¥n"
end
```

ArrayOfDefinition 型が返されるので、それぞれについて Definition 型の要 素 entry_id (パスウェイのID) と definition (パスウェイのタイトル) を取 り出します（先の SSDB の例も、実は SSDBRelation 型の要素 genes_id1 や sw_score などを取り出していたのでした）。

最後の例は、大腸菌の遺伝子 b1002 と b2388 に対応するボックスに色を付け たパスウェイ eco00010 の画像を生成して、ファイルに保存する例です。

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

require 'bio'

serv = Bio::KEGG::API.new

genes = ["eco:b1002", "eco:b2388"]
url = serv.mark_pathway_by_objects("path:eco00010", genes)

puts url

# BioRuby の場合、画像を取得して保存するのに save_image メソッドが使える
serv.save_image(url, "filename.gif")
```

Perl の場合

Perl では、以下のライブラリを追加インストールしておく必要があります。

- SOAP::Lite

- MIME-Base64
- libwww-perl
- URI

以下、Ruby の最初の例と同じ処理を実行するサンプルコードです。

```
#!/usr/bin/env perl

use SOAP::Lite;

$wsdl = 'http://soap.genome.jp/KEGG.wsdl';

$serv = SOAP::Lite -> service($wsdl);

$start = 1;
$max_results = 5;

$top5 = $serv->get_best_neighbors_by_gene('eco:b0002', $start, $max_results);

foreach $hit (@{$top5}) {
  print "$hit->{genes_id1}¥t$hit->{genes_id2}¥t$hit->{sw_score}¥n";
}
```

同じく、大腸菌の KEGG パスウェイのリストを返す例です。

```
#!/usr/bin/env perl

use SOAP::Lite;

$wsdl = 'http://soap.genome.jp/KEGG.wsdl';

$results = SOAP::Lite
             -> service($wsdl)
             -> list_pathways("eco");

foreach $path (@{$result}) {
  print "$path->{entry_id}¥t$path->{definition}¥n";
}
```

SOAP::Lite では引数に配列を渡す時には、

```
SOAP::Data->type(array => [value1, value2, .. ])
```

のように変換する必要があるので注意が必要です。たとえばパスウェイへの色 づけで遺伝子のリストを渡す場合は、

```
#!/usr/bin/env perl

use SOAP::Lite;

$wsdl = 'http://soap.genome.jp/KEGG.wsdl';

$serv = SOAP::Lite -> service($wsdl);

$genes = SOAP::Data->type(array => ["eco:b1002", "eco:b2388"]);

$result = $serv -> mark_pathway_by_objects("path:eco00010", $genes);

print $result;
```

のようになります。

Python の場合

Python では以下のライブラリを追加インストールしておく必要があります。

- SOAPpy

また、SOAPpy が依存しているいくつかのパッケージ (fpconst, PyXML など) も 必要になります。

以下、KEGG/PATHWAY の 00020 番のパスウェイに載っている大腸菌の遺伝子を リストで返すサンプルコードです。

```
#!/usr/bin/env python

from SOAPpy import WSDL

wsdl = 'http://soap.genome.jp/KEGG.wsdl'
serv = WSDL.Proxy(wsdl)
```

```
results = serv.get_genes_by_pathway('path:eco00020')
print results
```

Java の場合

Java では Apache Axis ライブラリの axis-1.2alpha より新しいバージョン (axis-1_1 ではうまく動きません）を入手して、必要な jar ファイルを適切 なディレクトリに置いておく必要があります。

- Apache Axis

たとえば Apache Axis バージョン axis-1_2beta のバイナリ配布の場合、 axis-1_2beta/lib 以下にある jar ファイルをインストール先のディレクト リにコピーします。

```
% cp axis-1_2beta/lib/* /path/to/lib/
```

以下のように実行して WSDL から KEGG API 用のクラスを自動生成します。 また、生成されたファイルの不具合を直すために、axisfix.pl スクリプトを入手しておきます。

```
% java -classpath /path/to/lib/axis.jar:/path/to/lib/jaxrpc.jar:/path/to/lib/commons-logging.ja
% perl -i axisfix.pl keggapi/KEGGBindingStub.java
% javac -classpath /path/to/lib/axis.jar:/path/to/lib/jaxrpc.jar:/path/to/lib/wsdl4j.jar:. kegg
% jar cvf keggapi.jar keggapi/*
% javadoc -classpath /path/to/lib/axis.jar:/path/to/lib/jaxrpc.jar -d keggapi_javadoc keggapi/*
```

javadoc の英語版が必要な場合は javadoc に -locale en_US オプションを つけて実行します。

以下は、Python の例と同様に、指定した KEGG/PATHWAY に載っている遺伝子の リストを表示するサンプルコードです。

```
import keggapi.*;

class GetGenesByPathway {
        public static void main(String[] args) throws Exception {
                KEGGLocator  locator = new KEGGLocator();
                KEGGPortType serv    = locator.getKEGGPort();

                String   query   = args[0];
                String[] results = serv.get_genes_by_pathway(query);

                for (int i = 0; i < results.length; i++) {
                        System.out.println(results[i]);
                }
        }
}
```

次は、SSDBRelation 型の配列が戻ってくる例です。

```
import keggapi.*;

class GetBestNeighborsByGene {
        public static void main(String[] args) throws Exception {
                KEGGLocator    locator  = new KEGGLocator();
                KEGGPortType   serv     = locator.getKEGGPort();

                String         query    = args[0];
                SSDBRelation[] results  = null;

                results = serv.get_best_neighbors_by_gene(query, 1, 50);

                for (int i = 0; i < results.length; i++) {
                        String gene1  = results[i].getGenes_id1();
                        String gene2  = results[i].getGenes_id2();
                        int    score  = results[i].getSw_score();
                        System.out.println(gene1 + "¥t" + gene2 + "¥t" + score);
                }
        }
}
```

このプログラムは以下のように -classpath オプションに keggapi.jar ファ イルも加えてコンパイル、実行します。

```
% javac -classpath /path/to/lib/axis.jar:/path/to/lib/jaxrpc.jar:/path/to/lib/wsdl4j.jar:/path/

% java -classpath /path/to/lib/axis.jar:/path/to/lib/jaxrpc.jar:/path/to/lib/commons-logging.ja
```

環境変数 CLASSPATH を指定しておくと、長いオプションを毎回書く必要がな くなります。

bash または zsh の場合：

```
% for i in /path/to/lib/*.jar
do
  CLASSPATH="${CLASSPATH}:${i}"
done
% export CLASSPATH
```

tcsh の場合：

```
% foreach i ( /path/to/lib/*.jar )
  setenv CLASSPATH ${CLASSPATH}:${i}
end
```

他の戻り値と型ごとの値の取り出し方などについては、WSDL2Java により生成 された以下のドキュメントを参照してください。

- <URL:http://www.genome.jp/kegg/soap/doc/keggapi_javadoc_ja/>

KEGG API リファレンス

以下では、KEGG API を使うのに必要な情報と全てのメソッドを解説します。

WSDL ファイル

SOAP では、サーバがどのようなメソッドを持っているか知っておく必要が ありますが、WSDL を使うとこの手順を自動化できます。WSDL ファイルを 取得してクライアントドライバを生成するところまで、通常は SOAP/WSDL の ライブラリが処理してくれるはずです。KEGG API の WSDL ファイルは以下の URL にあります。

- <URL:http://soap.genome.jp/KEGG.wsdl>

用語の凡例

以下の解説で出てくる KEGG 関連用語の説明をしておきます。

- org は KEGG に含まれている生物種をそれぞれ３文字コードで 表記したもので、eco が大腸菌、sce が出芽酵母などとなっています。 ３文字コードのリストは以下のページにあります。
  <URL:http://www.genome.jp/kegg/kegg2.html#genes>
- db は GenomeNet で提供されているデータベース名です。データベース名の リストについては list_databases メソッドを参照してください。
- entry_id は db_name とエントリ名を ':' で結合した全てのデータベース間で ユニークな ID です。たとえば embl:J00231 で EMBL のエントリ J00231 を 指します。entry_id は、以下の genes_id, enzyme_id, compound_id, reaction_id, pathway_id, motif_id などを含みます。
- genes_id は keggorg と遺伝子名を ':' で結合した KEGG の遺伝子 ID です。 eco:b0001 は大腸菌の遺伝子 b0001 を指します。
- enzyme_id は ec: をつけた酵素番号の ID です。ec:1.1.1.1 は酵素番号 1.1.1.1 の酵素であるアルコール・デヒドロゲナーゼを指します。
- compound_id は cpd: をつけた化合物の ID です。cpd:C00158 は化合物番号 C00158 の化合物であるクエン酸を指します。
- reaction_id は REACTION データベースのエントリ番号で、rn:R00959 は リアクション番号 R00959 の反応 (cpd:C00103 と cpd:00668 間の変換) を 指します。
- pathway_id は KEGG/PATHWAY データベースのパスウェイ番号で、パスウェイ 番号のプレフィックスが map の場合はリファレンスパスウェイを、keggorg の 場合はその生物種の持つ遺伝子の載ったパスウェイを表します。例えば path:map00020 はリファレンスパスウェイの 00020 番を、path:eco00020 は 大腸菌のパスウェイ 00020 番を指します。
- motif_id はモチーフデータベースのエントリ名で、pf:DnaJ で Pfam のエントリ DnaJ を指します。'pf' の他、'ps' で PROSITE, 'bl' で BLOCS, 'pr' で PRINTS, 'pd' で PRODOM を指します。
- ko_id は KO (KEGG Orthology) データベースのエントリ番号で、ko:K02598 は KO 番号 K02598 の nitrite transporter NirC のオーソログガスな遺伝子 グループを指します。
- start, max_results は一度に返ってくる結果の数を指定するオプションで、 start 番目から max_results 個の結果を受け取ります。全ての結果を得るには start = start + max_results として空の配列が返ってくるまで繰り返し メソッドを呼びます。
- fg_color_list はパスウェイへの色づけでオブジェクトごとに文字や枠線の 色を指定する配列です。
- fg_color_list はパスウェイへの色づけでオブジェクトごとに背景の 色を指定する配列です。

戻り値のデータ型

KEGG API のメソッドは文字列など単純な値を返すものだけでなく、複雑なデータ 構造を持った値を返す場合もあり、そのためのデータ型が定義されています。

SSDBRelation 型

SSDB データベースの各フィールドに対応する値が入った配列です。

```
genes_id1          クエリーの genes_id (string)
genes_id2          ターゲットの genes_id (string)
sw_score           genes_id1 と genes_id2 間の Smith-Waterman スコア (int)
bit_score          genes_id1 と genes_id2 間の bit スコア (float)
identity           genes_id1 と genes_id2 間の アイデンティティ (float)
overlap            genes_id1 と genes_id2 のオーバーラップ領域の長さ (int)
```

```
start_position1    genes_id1 のアライメントの開始残基位置 (int)
end_position1      genes_id1 のアライメントの終端残基位置 (int)
start_position2    genes_id2 のアライメントの開始残基位置 (int)
end_position2      genes_id2 のアライメントの終端残基位置 (int)
best_flag_1to2     genes_id1 から見て genes_id2 がベストヒットか (boolean)
best_flag_2to1     genes_id2 から見て genes_id1 がベストヒットか (boolean)
definition1        genes_id1 のデフィニション文字列 (string)
definition2        genes_id2 のデフィニション文字列 (string)
length1            genes_id1 のアミノ酸配列の長さ (int)
length2            genes_id2 のアミノ酸配列の長さ (int)
```

ArrayOfSSDBRelation 型

複数の SSDBRelation 型データを含む配列です。

MotifResult 型

```
motif_id           モチーフデータベースのエントリ ID (string)
definition         モチーフのデフィニション (string)
genes_id           モチーフを持っている遺伝子の genes_id (string)
start_position     モチーフの開始残基位置 (int)
end_position       モチーフの終了残基位置 (int)
score              モチーフ (PROSITE Profile, TIGRFAM) のスコア (float)
evalue             モチーフ (Pfam) の E-value (double)
```

注：score と evalue のうち、対応する値が無いものについては -1 を返します。

ArrayOfMotifResult 型

複数の MotifResult 型データを含む配列です。

Definition 型

```
entry_id           データベースエントリーのID (string)
definition         エントリーのデフィニション情報 (string)
```

ArrayOfDefinition 型

複数の Definitioin 型データを含む配列です。

LinkDBRelation 型

```
entry_id1          データベースのエントリ ID (string)
entry_id2          データベースのエントリ ID (string)
type               "direct" または "indirect" のリンクの種類 (string)
path               リンクの経路 (string)
```

ArrayOfLinkDBRelation 型

複数の LinkDBRelation 型データを含む配列です。

メソッド一覧

以下、KEGG API の全メソッドのリストです。メソッドにはメタ情報を返すものと 各データベースに対するものがあります。現在、KEGG にあるデータベースのうち KEGG API の対象となっているものは SSDB, PATHWAY, GENES, LIGAND です。これ 以外のデータベースへの対応やメソッドの追加も順次おこなう予定です。

以下の例では、引数などが Ruby 言語の表記に倣って書かれていますが、実際の 引数（特にリストの渡し方など）は使用する言語によって異なる可能性があります。

メタ情報

最新のデータベース情報などを返すためのメソッドです。

list_databases

KEGG を提供しているゲノムネットで現在利用できるデータベースの一覧を返します。

戻値：

```
ArrayOfDefinition
```

list_organisms

現在 KEGG に含まれている生物種 (org) のリストを返します。

戻値：

```
ArrayOfDefinition
```

list_pathways(*org*)

現在 KEGG に含まれている指定した生物のパスウェイのリストを返します。引数に 'map' という文字列を与えるとリファレンスパスウェイのリストを返します。

戻値：

```
ArrayOfDefinition
```

DBGET

DBGET システムに対するメソッドの一覧です。DBGET について詳しくは以下の ページを参照してください。

- <URL:http://www.genome.jp/dbget/dbget_manual.html>

binfo(*string*)

指定したデータベースのエントリ数や更新日など詳しい最新情報を返します。 'all' を渡すと利用可能な全てのデータベースの情報を返します。 binfo コマンドへの引数を文字列で渡します。

戻値：

```
string
```

例：

```
# GenBank データベースの最新情報
binfo('gb')
# 全てのデータベースの最新情報
binfo('all')
```

bfind(*string*)

DBGET の bfind コマンドへのラッパーです。キーワードによりエントリを 検索することができます。一度に与えられるキーワードの数は 100 個以下に 制限されています。

戻値：

```
string
```

例：

```
# デフィニションに E-cadherin と human を持つ GenBank のエントリを検索
bfind("gb E-cadherin human")
```

bget(*string*)

指定した entry_id のエントリを返します。GENES の遺伝子エントリをはじめ、 ゲノムネットの DBGET システムで提供されている様々なデータベース (list_databases を参照) のエントリを全て取得できます。bget コマンドへの コマンドラインオプションを文字列で渡します。一度に取得できるエントリの 数は 100 個以下に制限されています。

戻値：

```
string
```

例：

```
# 複数のエントリを取得
bget("eco:b0002 bsu:BG10065 cpd:C00209")
# FASTA フォーマットのアミノ酸配列を取得
bget("-f -n a eco:b0002 bsu:BG10065")
# FASTA フォーマットの塩基配列を取得
bget("-f -n n eco:b0002 hin:tRNA-Cys-1")
```

btit(*string*)

DBGET の btit コマンドへのラッパーです。指定したエントリの ID に対応す るデフィニションを返します。一度に与えられるエントリの数は 100 個以下に 制限されています。

戻値：

```
string
```

例：

```
# これら４つの遺伝子のデフィニションを検索
btit("hsa:1798 mmu:13478 dme:CG5287-PA cel:Y60A3A.14")
```

LinkDB

get_linkdb_by_entry(*entry_id*, *db*, *start*, *max_results*)

指定した entry_id から直接または間接的にリンクされているエントリの経路を、 db で指定したデータベースにたどれるまで検索します。

戻値：

```
ArrayOfLinkDBRelation
```

例：

```
# E. coli の遺伝子 b0002 からリンクのたどれる KEGG/PATHWAY のエントリを検索
get_linkdb_by_entry('eco:b0002', 'pathway', 1, 10)
get_linkdb_by_entry('eco:b0002', 'pathway', 11, 10)
```

SSDB

SSDB データベースに対するメソッドの一覧です。SSDB は KEGG/GENES に含まれる 全生物種・全遺伝子間で ssearch を用いた Smith-Waterman アルゴリズムに よる検索を行った結果と、全遺伝子のモチーフ検索結果を登録したデータベースで、 時間のかかる計算があらかじめ終わっているため高速な検索が可能になっています。

KEGG がゲノム配列の決まった生物種を中心に対象としていることと、Smith- Waterman スコアによる比較ができることからオーソログやパラログ関係にある 遺伝子の探索や生物種固有の遺伝子の検索をはじめ様々な応用が考えられます。

SSDB データベースについて詳しくは以下のページを参照してください。

- <URL:http://www.genome.jp/kegg/ssdb/>

get_neighbors_by_gene(*genes_id*, *org*, *start*, *max_results*)

指定した genes_id の遺伝子にホモロジーのある全遺伝子を指定した生物から 検索します。また org に 'all' を指定すると全生物種から検索します。

戻値：

```
ArrayOfSSDBRelation
```

例：

```
# 大腸菌の遺伝子 b0002 に相同性のある遺伝子を全て検索して、結果の１番
# から１０番目までを返します
get_neighbors_by_gene('eco:b0002', 'all' 1, 10)
# 次の１０個を start = start + max_results として取ります
get_neighbors_by_gene('eco:b0002', 'all' 11, 10)
```

get_best_best_neighbors_by_gene(*genes_id*, *start*, *max_results*)

クエリとターゲットが best-best の関係にある遺伝子だけを検索します。

戻値：

```
ArrayOfSSDBRelation
```

例：

```
# 大腸菌の遺伝子 b0002 から全生物種で best-best の関係にある遺伝子
get_best_best_neighbors_by_gene('eco:b0002', 1, 10)
get_best_best_neighbors_by_gene('eco:b0002', 11, 10)
```

get_best_neighbors_by_gene(*genes_id*, *start*, *max_results*)

クエリから見てベストヒットの関係にある遺伝子だけを検索します。

戻値：

```
ArrayOfSSDBRelation
```

例：

```
# 大腸菌の遺伝子 b0002 から全生物種で best neighbor の関係にある遺伝子
get_best_neighbors_by_gene('eco:b0002', 1, 10)
get_best_neighbors_by_gene('eco:b0002', 11, 10)
```

get_reverse_best_neighbors_by_gene(*genes_id, start, max_results*)

ターゲット側の生物種から見てクエリがベストヒットとなる遺伝子を検索します。

戻値：

```
ArrayOfSSDBRelation
```

例：

```
# 大腸菌の遺伝子 b0002 が reverse best neighbor の関係にある遺伝子
get_reverse_best_neighbors_by_gene('eco:b0002', 1, 10)
get_reverse_best_neighbors_by_gene('eco:b0002', 11, 10)
```

get_paralogs_by_gene(*genes_id, start, max_results*)

クエリと同じ生物種内でパラログ遺伝子を検索します。

戻値：

```
ArrayOfSSDBRelation
```

例：

```
# 大腸菌の遺伝子 b0002 とパラログの関係にある遺伝子
get_paralogs_by_gene('eco:b0002', 1, 10)
get_paralogs_by_gene('eco:b0002', 11, 10)
```

get_similarity_between_genes(*genes_id1, genes_id2*)

指定した２つの遺伝子間の Smith-Waterman スコアを含むデータを返します。

戻値：

```
SSDBRelation
```

例：

```
# 大腸菌の b0002 遺伝子と b3940 遺伝子間のスコアやアライメント領域を得る
get_similarity_between_genes('eco:b0002', 'eco:b3940')
```

Motif

get_motifs_by_gene(*genes_id, db*)

指定した遺伝子に存在するモチーフのリストを返します。モチーフデータベース のリストには、Pfam (pfam), TIGRFAM (tfam), PROSITE pattern (pspt), PROSITE profile (pspf) またはこれら全て (all) を指定出来ます。

戻値：

```
ArrayOfMotifResult
```

例：

```
# 大腸菌の遺伝子 b0002 持つPfamモチーフのリスト
get_motifs_by_gene('eco:b0002', 'pfam')
```

get_genes_by_motifs(*motif_id_list, start, max_results*)

指定したモチーフを持つ遺伝子を検索します。

戻値：

```
ArrayOfDefinition
```

例：

```
# Pfam の DnaJ と Prosite の DNAJ_2 にヒットする遺伝子を検索
list = ['pf:DnaJ', 'ps:DNAJ_2']
get_genes_by_motifs(list, 1, 10)
get_genes_by_motifs(list, 11, 10)
```

KO, OC, PC

KO (KEGG orthology), OC (KEGG ortholog cluster), PC (KEGG paralog cluster) の 情報を得るためのメソッドです。KO はキュレーションされたオーソログ遺伝子群、 OC と PC は機械的にクラスタリングされた相同性のある遺伝子群のデータベースです。

get_ko_by_gene(*genes_id*)

指定した遺伝子にアサインされている KO のエントリ番号を全て返します。

戻値：

```
ArrayOfstring
```

例：

```
# eco:b0002 遺伝子にアサインされている KO のリスト
get_ko_by_gene('eco:b0002')
```

get_ko_members(*ko_id*)

指定した ko_id の KO エントリに含まれる遺伝子のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# KO 番号 K02208 のアサインされている遺伝子のリスト
get_ko_by_gene('ko:K02598')
```

get_oc_members_by_gene(*genes_id*, *start*, *max_results*)

指定した遺伝子と同じ OC に属する遺伝子のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# eco:b0002 遺伝子と同じオーソログクラスターに含まれる遺伝子のリスト
get_oc_members_by_gene('eco:b0002', 1, 10)
get_oc_members_by_gene('eco:b0002', 11, 10)
```

get_pc_members_by_gene(*genes_id*, *start*, *max_results*)

指定した遺伝子と同じ PC に属する遺伝子のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# eco:b0002 遺伝子と同じパラログクラスターに含まれる遺伝子のリスト
get_pc_members_by_gene('eco:b0002', 1, 10)
get_pc_members_by_gene('eco:b0002', 11, 10)
```

PATHWAY

PATHWAY データベースに対するメソッドの一覧です。PATHWAY データベースに ついて詳しくは以下のページを参照してください。

- <URL:http://www.genome.jp/kegg/kegg2.html#pathway>

パスウェイへの色づけ

mark_pathway_by_objects(*pathway_id*, *object_id_list*)

指定した生物種で、与えられたパスウェイマップの与えられたオブジェクト （遺伝子、化合物、酵素番号）の対応する枠に色をつけた画像を生成、 画像の URL を返します。

戻値：

```
string
```

例：

```
# 大腸菌のパスウェイ path:eco00260 上の遺伝子 eco:b0002 と Homoserine
# の cpd:C00263 に対応するボックスを赤く着色した画像の URL
obj_list = ['eco:b0002', 'cpd:C00263']
mark_pathway_by_objects('path:eco00260', obj_list)
```

color_pathway_by_objects(*pathway_id*, *object_id_list*, *fg_color_list*, *bg_color_list*)

指定したパスウェイの与えられたオブジェクト（遺伝子、化合物、酵素）に対し、 文字と枠に fg_color_list で指定した色、背景に bg_color_list で指定した色を つけた画像を生成、画像の URL を返します。object_id_list と fg_color_list, bg_color_list の要素の数と順番を揃えるように注意する必要があります。

戻値：

```
string
```

例：

```
# パスウェイ path:eco00053 上に載っている大腸菌の遺伝子 eco:b0207 を
# 背景が赤、文字と枠を青で着色し、eco:b1300 の背景を黄色、文字と枠を緑で
# 着色した画像の URL を返します。
obj_list = ['eco:b0207', 'eco:b1300']
fg_list  = ['blue', '#00ff00']
bg_list  = ['#ff0000', 'yellow']
color_pathway_by_objects('path:eco00053', obj_list, fg_list, bg_list)
```

パスウェイ上のオブジェクト検索

get_genes_by_pathway(*pathway_id*)

指定したパスウェイ上に載っている遺伝子のリストを返します。生物種名は pathway_id に含まれる keggorg で指定します。

戻値：

```
ArrayOfstring
```

例：

```
# 大腸菌のパスウェイ 00020 番に載っている遺伝子のリスト
get_genes_by_pathway('path:eco00020')
```

get_enzymes_by_pathway(*pathway_id*)

指定したパスウェイに載っている酵素番号のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# パスウェイ 00020 番に載っている酵素番号のリスト
get_enzymes_by_pathway('path:eco00020')
```

get_compounds_by_pathway(*pathway_id*)

指定したパスウェイに載っている化合物のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# パスウェイ 00020 に載っている化合物のリスト
get_compounds_by_pathway('path:eco00020')
```

get_reactions_by_pathway(*pathway_id*)

指定したパスウェイに載っているリアクション番号のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# パスウェイ 00260 番に載っているリアクションのリスト
get_reactions_by_pathways('path:map00260')
```

オブジェクトからパスウェイ検索

get_pathways_by_genes(*genes_id_list*)

指定した遺伝子が全て載っているパスウェイのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 大腸菌の遺伝子 b0077 と b0078 が両方載っているパスウェイのリスト
get_pathways_by_genes(['eco:b0077', 'eco:b0078'])
```

get_pathways_by_enzymes(*enzyme_id_list*)

指定した酵素番号が全て載っているパスウェイのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 酵素番号 1.3.99.1 の酵素が載っているパスウェイのリスト
get_pathways_by_enzymes(['ec:1.3.99.1'])
```

get_pathways_by_compounds(*compound_id_list*)

指定した化合物が全て載っているパスウェイのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 化合物 C00033 と C00158 が両方載っているパスウェイのリスト
get_pathways_by_compounds(['cpd:C00033', 'cpd:C00158'])
```

get_pathways_by_reactions(*reaction_id_list*)

指定したリアクション番号が全て載っているパスウェイのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# リアクション番号 rn:R00959, rn:R02740, rn:R00960, rn:R01786 の全ての
# 反応を含むパスウェイのリスト
get_pathways_by_reactions(['rn:R00959', 'rn:R02740', 'rn:R00960', 'rn:R01786'])
```

パスウェイ間の関係

get_linked_pathways(*pathway_id*)

指定したパスウェイ番号のパスウェイからリンクされているパスウェイの リストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# パスウェイ path:eco00620 からリンクされているパスウェイのリスト
get_linked_pathways('path:eco00620')
```

遺伝子と酵素番号の関係

get_genes_by_enzyme(*enzyme_id, org*)

対象生物種において、指定した酵素番号を持つ遺伝子のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 酵素番号 1.1.1.1 を持つ大腸菌の遺伝子のリスト
get_genes_by_enzyme('ec:1.1.1.1', 'eco')
```

get_enzymes_by_gene(*genes_id*)

指定した遺伝子に対応する酵素番号のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 大腸菌遺伝子 'eco:b0002' の酵素番号のリスト
get_enzymes_by_gene(eco:b0002)
```

酵素、化合物、リアクションの関係

get_enzymes_by_compound(*compound_id*)

指定した化合物番号に対応する酵素番号のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 化合物 'cpd:C00345' の代謝に関わる酵素のリスト
get_enzymes_by_compound('cpd:C00345')
```

get_enzymes_by_reaction(*reaction_id*)

指定したリアクション番号に対応する酵素番号のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# リアクション番号 R00100 を持つ酵素のリスト
get_enzymes_by_reaction('rn:R00100')
```

get_compounds_by_enzyme(*enzyme_id*)

指定した酵素番号に対応する化合物のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 酵素番号 'ec:2.7.1.12' の代謝に関わる化合物のリスト
get_compounds_by_enzyme('ec:2.7.1.12')
```

get_compounds_by_reaction(*reaction_id*)

指定したリアクションに対応する化合物のリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# リアクション番号 'rn:R00100' の反応に関わる化合物のリスト
get_compounds_by_reaction('rn:R00100')
```

get_reactions_by_enzyme(*enzyme_id*)

指定した酵素番号に対応するリアクションのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 酵素番号 'ec:2.7.1.12' の反応に関わるリアクション番号のリスト
get_reactions_by_enzyme('ec:2.7.1.12')
```

get_reactions_by_compound(*compound_id*)

指定した化合物に対応するリアクションのリストを返します。

戻値：

```
ArrayOfstring
```

例：

```
# 化合物 'cpd:C00199' の触媒反応に関わるリアクション番号のリスト
get_reactions_by_compound('cpd:C00199')
```

GENES

GENES データベースに対するメソッドの一覧です。GENES データベースについて 詳しくは以下のページを参照してください。

- <URL:http://www.genome.jp/kegg/kegg2.html#genes>

get_genes_by_organism(*org*, *start*, *max_results*)

指定した生物種の全 GENES エントリのうち、start 番目から max_results 分の 結果を返します。

戻値：

```
ArrayOfstring
```

例：

```
# インフルエンザ菌の遺伝子リストを 100 個ずつ得る
get_genes_by_organism('hin', 1, 100)
get_genes_by_organism('hin', 101, 100)
```

GENOME

GENOMES データベースに対するメソッドの一覧です。GENOME データベースについて 詳しくは以下のページを参照してください。

- <URL:http://www.genome.jp/kegg/kegg2.html#genome>

`get_number_of_genes_by_organism(org)`

指定した生物種が持つ遺伝子数を返します。

戻値：

```
int
```

例：

```
# 大腸菌が持つ遺伝子の数
get_number_of_genes_by_organism('eco')
```

Notes

Last updated: July 12, 2004

This document is written and maintained by Toshiaki Katayama.