2005/3/3 国立遺伝学研究所共同研究会「生物情報資源の相互運用性」

# スクリプト言語による ウェブサービスの利用

東京大学医科学研究所ヒトゲノム解析センター
片山俊明 <k@bioruby.org>

# ウェブサービスとは

- ウェブページを公開している
  - たぶん誤用
- フォームを使って CGI による DB 検索などが可能
  - ウェブ「で」サービスを提供している
- REST --- サービスを URI で規定
  - REpresentational State Transfer
- SOAP --- RPC, オブジェクトのやりとり
  - XML RPC
    - 直接 XML を扱う Remote Procedure Call
  - SOAP
    - Service Oriented Architecture Protocol
  - SOAP + WSDL
    - Web Service Description Language
  - SOAP + WSDL + UDDI
    - Universal Description, Discovery, and Integration

# (狭義の)ウェブサービスとは

- ウェブページを公開している
  - たぶん誤用
- フォームを使って CGI による DB 検索などが可能
  - ウェブ「で」サービスを提供している
- REST --- サービスを URI で規定
  - REpresentational State Transfer
- SOAP --- RPC, オブジェクトのやりとり
  - XML RPC
    - 直接 XML を扱う Remote Procedure Call
  - SOAP
    - Service Oriented Architecture Protocol
  - SOAP + WSDL
    - Web Service Description Language
  - SOAP + WSDL + UDDI
    - Universal Description, Discovery, and Integration

# バイオインフォのウェブサービス

- DBGET, dbfetch (SRS), BioFetch --- REST
- Entrez E-Utilities (NCBI) --- REST
- BioDAS (WormBase, Ensemblなど) --- REST
- XML Central of DDBJ (遺伝研) --- SOAP/WSDL
- KEGG API (ゲノムネット) --- SOAP/WSDL
- EBI Web Services --- SOAP/WSDL
- ESOAP (SOAP版 E-Utils) --- SOAP/WSDL

# RESTの例

- BioFetch
  - EMBL から2エントリ取得

    http://www.ebi.ac.uk/cgi-bin/dbfetch?db=EMBL&id=J00231,BUM

  - ↑を FASTA フォーマットで

    http://www.ebi.ac.uk/cgi-bin/dbfetch?db=EMBL&id=J00231,BUM&format=fasta

- BioDAS
  - WormBase からある領域の配列を取得

    http://www.wormbase.org/db/das/elegans/dna?segment=I:1,20000

  - WormBase から同じ領域のアノテーションを取得

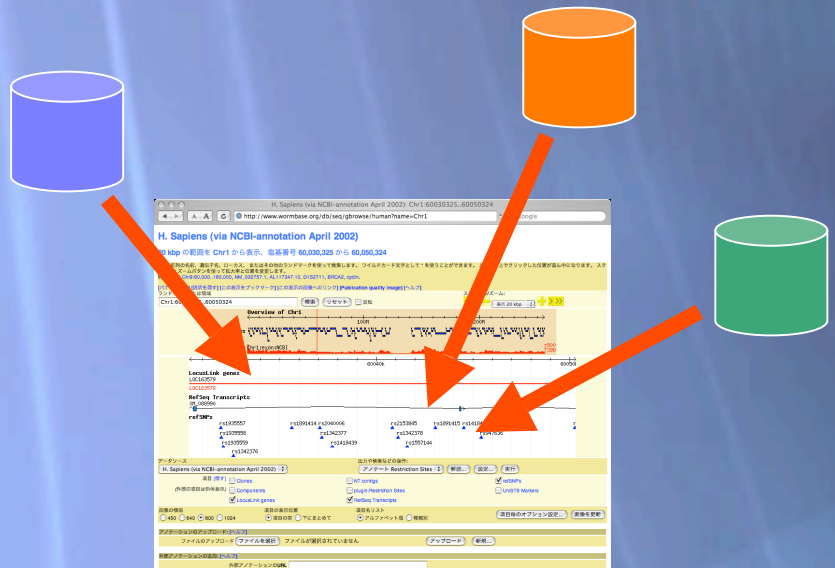
    http://www.wormbase.org/db/das/elegans/features?segment=I:1,20000

# WormBaseのDASデータ

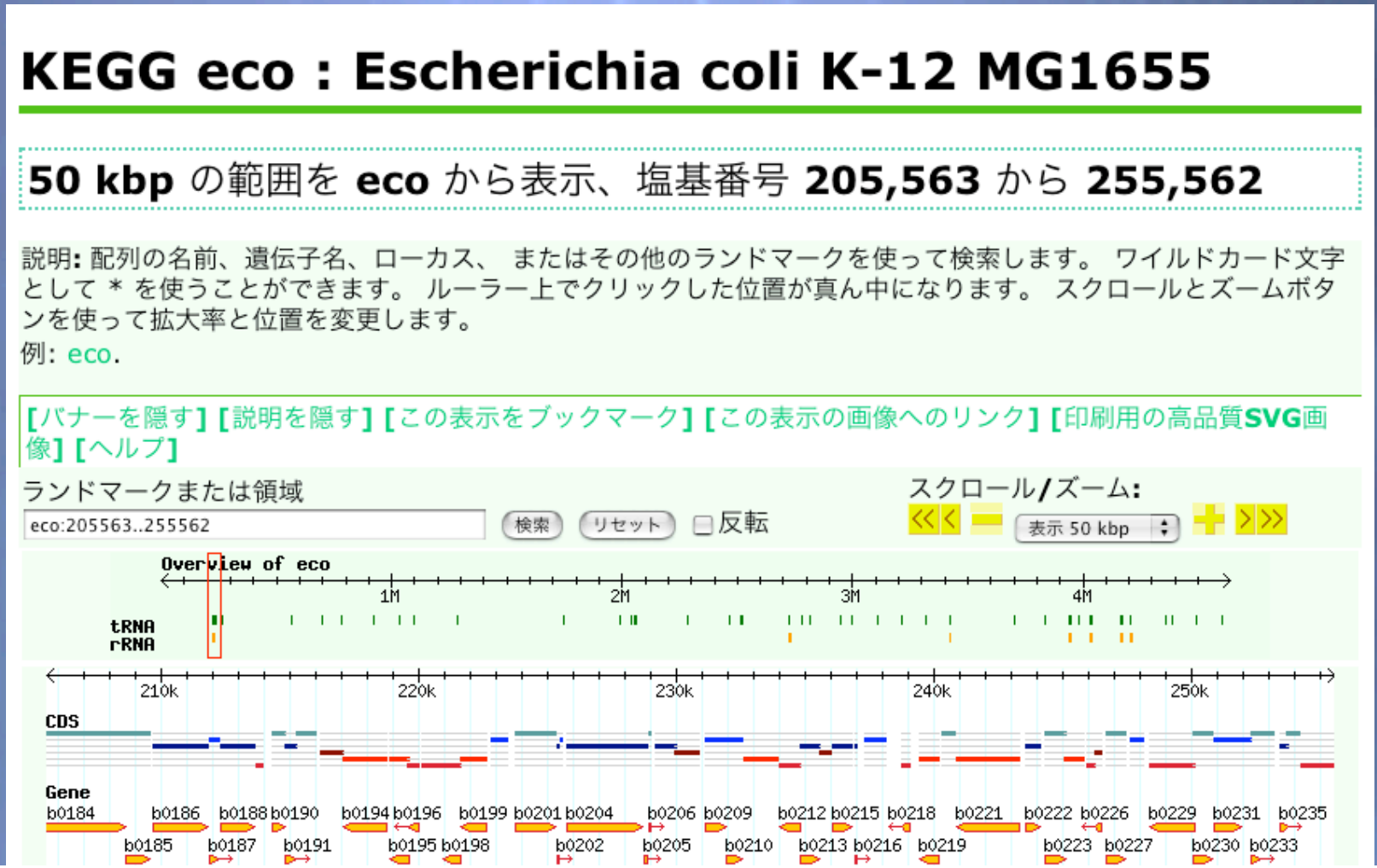
```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DASGFF SYSTEM "http://www.biodas.org/dtd/dasgff.dtd">
<DASGFF>
<GFF version="1.01" href="http://www.wormbase.org/db/das/elegans/features?segment=I%3A1%2C20000">
<SEGMENT id="I" start="1" stop="20000" version="1.0">
  <FEATURE id="Sequence:yk582g6.5/241592" label="yk582g6.5">
    <TYPE id="EST_match:BLAT_EST_OTHER" category="miscellaneous">EST_match:BLAT_EST_OTHER</TYPE>
    <METHOD id="EST_match">EST_match</METHOD>
    <START>1</START>
    <END>22</END>
    <SCORE>14.2</SCORE>
    <ORIENTATION>+</ORIENTATION>
    <PHASE>0</PHASE>
    <LINK href="http://www.wormbase.org/db/get?name=yk582g6.5;class=Sequence">yk582g6.5</LINK>
    <TARGET id="yk582g6.5" start="284" stop="305" />
    <GROUP id="Sequence:yk582g6.5" type="Sequence" />
  </FEATURE>
  <FEATURE id="Sequence:yk585b5.5/722458" label="yk585b5.5">
    <TYPE id="EST_match:BLAT_EST_OTHER" category="miscellaneous">EST_match:BLAT_EST_OTHER</TYPE>
    <METHOD id="EST_match">EST_match</METHOD>
    <START>1</START>
    <END>50</END>
    <SCORE>12.8</SCORE>
    <ORIENTATION>-</ORIENTATION>
    <PHASE>0</PHASE>
    <LINK href="http://www.wormbase.org/db/get?name=yk585b5.5;class=Sequence">yk585b5.5</LINK>
    <TARGET id="yk585b5.5" start="119" stop="168" />
    <GROUP id="Sequence:yk585b5.5" type="Sequence" />
  </FEATURE>
  <FEATURE id="161762" label="inverted_repeat:inverted">
    <TYPE id="inverted_repeat:inverted" category="miscellaneous">inverted_repeat:inverted</TYPE>
    <METHOD id="inverted_repeat">inverted_repeat</METHOD>
```

# DASとは

- Distributed Annotation System
- ゲノムアノテーションについて、REST なURI と XML (DTD) を決めたもの
- ゲノムデータベース間での相互運用性
  - Ensembl
  - UCSC
  - WormBase
  - FlyBase
  - KEGG DAS etc.



# KEGG DAS - GBrowse



# KEGG DASでのデータ取得例

- 先ほどの画像

  http://das.hgc.jp/cgi-bin/gbrowse/eco?name=eco:205563..255562

- 同じ内容の DAS データ

  http://das.hgc.jp/cgi-bin/das/eco/features?segment=eco:205563,255562

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DASGFF SYSTEM "http://www.biodas.org/dtd/dasgff.dtd">
<DASGFF>
<GFF version="1.01" href="http://das.hgc.jp/cgi-bin/das/eco/features?segment=eco%3A205563%2C255562">
<SEGMENT id="eco" start="205563" stop="255562" version="1.0">
  <FEATURE id="EC:1.1.1.-/649" label="1.1.1.-">
    <TYPE id="enzyme:KEGG" category="enzyme">enzyme:KEGG</TYPE>
    <METHOD id="enzyme">enzyme</METHOD>
    <START>229167</START>
    <END>229970</END>
    <SCORE>-</SCORE>
    <ORIENTATION>+</ORIENTATION>
    <PHASE>0</PHASE>
    <LINK href="http://www.genome.jp/dbget-bin/www_bget?EC:1.1.1.-">1.1.1.-</LINK>
    <GROUP id="EC:1.1.1.-" type="EC" />
  </FEATURE>
  <FEATURE id="EC:1.3.99.-/700" label="1.3.99.-">
    <TYPE id="enzyme:KEGG" category="enzyme">enzyme:KEGG</TYPE>
    <METHOD id="enzyme">enzyme</METHOD>
    <START>240859</START>
```

# BioRubyによるDASアクセス

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

require 'bio'

serv = Bio::DAS.new("http://das.hgc.jp/cgi-bin/")

# 大腸菌(eco)のゲノム領域 200563〜255562 について
segment = Bio::DAS::SEGMENT.region("eco", 205563, 255562)

# ゲノム DNA 配列の取得
results = serv.get_dna("eco", segment)
results.each do |dna|
  puts dna.sequence
end

# アノテーションの取得
results = serv.get_features("eco", segment)
results.segments.each do |segment|
  segment.features.each do |feature|
    puts feature.entry_id
    puts feature.start
  end
end
```

# RESTの利点

- URI のルールを覚えるだけなので単純
- ブラウザ、コマンドラインなどクライアントを選ばない、特別な準備が要らない
- サーバ側の開発にフォーマットや実装などの自由度が高い

# RESTの欠点

- サーバがデータをどのようなフォーマットで返してくるか不明
- XML が返ってきた場合でも、自前でパースして欲しいデータを抽出する必要がある

# SOAPの例

- KEGG API
  http://www.genome.jp/kegg/soap/
  - DBGET, GENES, パスウェイ解析
- XML Central of DDBJ
  http://xml.ddbj.nig.ac.jp/
  - DDBJ, 相同性検索, GIB, GTOP, PML etc.
- EBI Web Services
  http://www.ebi.ac.uk/Tools/webservices/
  - DBFetch, WU-BLAST, FASTA, InterProScan
- NCBI ESOAP
  http://eutils.ncbi.nlm.nih.gov/entrez/query/static/esoap_help.html
  - Entrez のインターフェイス (EFetch, ESearchなど)

# SOAPとは

- XMLのSOAPメッセージをサーバとやり取り
- サーバにリクエスト
  - get_genes_by_pathway("path:eco00010")
- 結果を得る
  - 遺伝子のリスト
    - "eco:b0114", "eco:b0115", "eco:b0116", ...

# WSDLとは

- XMLで記述された、サーバで提供されているメソッドとデータ型の一覧

```
<!-- color_pathway_by_objects -->
<message name="color_pathway_by_objectsRequest">
        <part name="pathway_id"         type="xsd:string"/>
        <part name="object_id_list"     type="typens:ArrayOfstring"/>
        <part name="fg_color_list"      type="typens:ArrayOfstring"/>
        <part name="bg_color_list"      type="typens:ArrayOfstring"/>
</message>

<message name="color_pathway_by_objectsResponse">
        <part name="return"     type="xsd:string"/>
</message>

<!-- Objects on the pathway -->
<!-- get_genes_by_pathway -->
<message name="get_genes_by_pathwayRequest">
        <part name="pathway_id" type="xsd:string"/>
</message>

<message name="get_genes_by_pathwayResponse">
        <part name="return"     type="typens:ArrayOfstring"/>
</message>

<!-- get_enzymes_by_pathway -->
<message name="get_enzymes_by_pathwayRequest">
```

# SOAP + WSDLの利点(1)

- クライアントの作成が簡単
  - WSDL ファイルからサーバで利用可能なサービスの一覧を得て、プログラムからライブラリ関数のように呼べるまでのセットアップがほぼ自動
- プログラム言語に依存しない
  - Perl, Python, Ruby, PHP, Java, C#

# たとえば Ruby の場合

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

require "soap/wsdlDriver"

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

# 大腸菌(eco)のあるパスウェイに載っている遺伝子一覧
puts serv.get_genes_by_pathway("path:eco00010")

# 大腸菌(eco)のパスウェイの一覧
list = serv.list_pathways("eco")
list.each do |path|
  puts "#{path.entry_id}¥t#{path.definition}¥n"
end
```

# たとえば Perl の場合

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

use SOAP::Lite;

$wsdl = "http://soap.genome.jp/KEGG.wsdl";
$serv = SOAP::Lite -> service($wsdl);

# 大腸菌(eco)のパスウェイの一覧
$list = $serv -> list_pathways("eco");

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

# 出力結果

```
# 大腸菌(eco)のあるパスウェイに載っている遺伝子一覧
get_genes_by_pathway("path:eco00010")

eco:b0114
eco:b0115
eco:b0116
eco:b0356
eco:b0688
  :

# 大腸菌(eco)のパスウェイの一覧
list_pathways("eco")

path:eco00010   Glycolysis / Gluconeogenesis - Escherichia coli K-12 MG1655
path:eco00020   Citrate cycle (TCA cycle) - Escherichia coli K-12 MG1655
path:eco00030   Pentose phosphate pathway - Escherichia coli K-12 MG1655
path:eco00040   Pentose and glucuronate interconversions - Escherichia coli K-12 MG1655
path:eco00051   Fructose and mannose metabolism - Escherichia coli K-12 MG1655
path:eco00052   Galactose metabolism - Escherichia coli K-12 MG1655
path:eco00053   Ascorbate and aldarate metabolism - Escherichia coli K-12 MG1655
path:eco00061   Fatty acid biosynthesis (path 1) - Escherichia coli K-12 MG1655
  :
```

# SOAPを使うための準備

- Ruby の場合
  - Ruby 1.8 で標準装備
    - つまり Ruby さえ入っていればすぐ使える
    - 最新の 1.8.2 以降をお勧め
- Perl の場合
  - CPAN から SOAP::Lite をインストール
    - その他必要なライブラリがいくつか
  - インストールしてしまえば手軽に使える

# ちなみにJavaの場合

- 環境設定
  - Apache Axisをインストール
  - WSDLファイルをダウンロード
  - org.apache.axis.wsdl.WSDL2Java
  - 作成したjarファイルをCLASSPATHに置く

# Javaによるアクセス

```
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]);
                }
        }
}
```

```
Axis には CLASSPATH が通っているとして:
% javac -classpath keggapi.jar   GetGenesByPathway.java
% java  -classpath keggapi.jar:. GetGenesByPathway path:eco00010
```

```
eco:b0114
eco:b0115
eco:b0116
  :
```

# SOAP + WSDLの利点(2)

- XML まわりの処理はライブラリが隠蔽
  - 実際にはメソッド呼び出し、戻ってきた結果はともに XML でやり取りされている
  - しかし、実行例を見て分かるようにユーザは意識する必要がない

# 実際に流れているXMLは？

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

require 'soap/wsdlDriver'

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

# 標準エラー出力に通信データを表示
serv.wiredump_dev = STDERR

# 大腸菌(eco)のあるパスウェイに載っている遺伝子一覧
puts serv.get_genes_by_pathway("path:eco00010")
```

# SOAPのXMLメッセージ

Wire dump:

opening connection to soap.genome.jp...
opened
<- "POST /keggapi/request_v3.2.cgi HTTP/1.1\r\nAccept: */*\r\nContent-Type: text/xml; charset=utf-8\r\nUser-Agent: SOAP4R/1.5.3-ruby1.8.2\r\nSoapaction: \"SOAP/KEGG#get_genes_by_pathway\"\r\nContent-Length: 336\r\nHost: soap.genome.jp\r\n\r\n"
<- "<?xml version=\"1.0\" encoding=\"utf-8\" ?>\n<env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/envelope/\"\n xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n <env:Body>\n <n1:get_genes_by_pathway xmlns:n1=\"SOAP/KEGG\">\n <pathway_id>path:eco00010</pathway_id>\n </n1:get_genes_by_pathway>\n </env:Body>\n</env:Envelope>"
-> "HTTP/1.1 200 OK\r\n"
-> "Date: Thu, 03 Mar 2005 02:02:19 GMT\r\n"
-> "Server: Apache/1.3.26 (Unix)\r\n"
-> "SOAPServer: SOAP::Lite/Perl/0.55\r\n"
-> "Content-Length: 2422\r\n"
-> "Content-Type: text/xml; charset=utf-8\r\n"
-> "\r\n"
reading 2422 bytes...
-> "<?xml version=\"1.0\" encoding=\"UTF-8\"?><SOAP-ENV:Envelope xmlns:SOAP-ENC=\"http://schemas.xmlsoap.org/soap/encoding/\" SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\" xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\" xmlns:xsi=\"http://www.w3.org/1999/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/1999/XMLSchema\"><SOAP-ENV:Body><namesp1:get_genes_by_pathwayResponse xmlns:namesp1=\"SOAP/KEGG\"><return SOAP-ENC:arrayType=\"xsd:string[42]\" xsi:type=\"SOAP-ENC:Array\"><item xsi:type=\"xsd:string\">eco:b0114</item><item xsi:type=\"xsd:string\">eco:b0115</item><item xsi:type=\"xsd:string\">eco:b0116</item><item xsi:type=\"xsd:string\">eco:b0356</item><item xsi:type=\"xsd:string\">eco:b0688</item><item xsi:type=\"xsd:string\">eco:b0755</item><item

# SOAP + WSDLの利点(3)

- まさにオブジェクトをやり取りしている感じ
  - Java や Ruby のインスタンスをそのまま SOAP オブジェクトにマッピング（を勝手にライブラリがやってくれる）
    - リストが返ってくるメソッドは結果を配列としてアクセス
    - 構造体のようなデータをそのまま受け渡す
  - データのモデル化がされている
    - さらにデータ（ベースエントリ）がフラットではなくオブジェクトとして返ってくればパースは要らなくなる

# データ構造

```
# 大腸菌(eco)のパスウェイの一覧
list = serv.list_pathways("eco")          # => ArrayOfDefinition
list.each do |path|                       # => Definition型オブジェクト
  puts "#{path.entry_id}¥t#{path.definition}¥n"
end
```

**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) |

```
<SOAP-ENV:Body><namesp1:list_pathwaysResponse xmlns:namesp1=\"SOAP/KEGG\">
<return SOAP-ENC:arrayType=\"namesp2:SOAPStruct[111]\" xsi:type=\"SOAP-ENC:Array\">
<item xsi:type=\"namesp2:SOAPStruct\">
 <definition xsi:type=\"xsd:string\">
  Glycolysis / Gluconeogenesis - Escherichia coli K-12 MG1655
 </definition>
 <entry_id xsi:type=\"xsd:string\">path:eco00010</entry_id>
</item>
<item xsi:type=\"namesp2:SOAPStruct\">
 <definition xsi:type=\"xsd:string\">
  Citrate cycle (TCA cycle) - Escherichia coli K-12 MG1655
 </definition>
 <entry_id xsi:type=\"xsd:string\">path:eco00020</entry_id>
</item>
  :
path:eco00010   Glycolysis / Gluconeogenesis - Escherichia coli K-12 MG1655
path:eco00020   Citrate cycle (TCA cycle) - Escherichia coli K-12 MG1655
path:eco00030   Pentose phosphate pathway - Escherichia coli K-12 MG1655
  :
```

# より複雑な構造も

**SSDBRelation 型**

| | |
|---|---|
| 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) |

```
# 大腸菌の遺伝子 b0002 とベストベストヒット関係にある遺伝子
list = serv.get_best_best_neighbors_by_gene("eco:b0002", 1, 100)
list.each do |hit|
  puts hit.genes_id1                    # => eco:b0002      eco:b0002
  puts hit.genes_id2                    # => ecj:JW0001     bsu:BG10350
  puts hit.sw_score                     # => 5283           561
end
```

# SOAPの欠点

- WSDLによっては、言語によって使えたり使えなかったり
- SAX なパーズ
  - ライブラリに依存する問題かも
- RESTの方がライトウェイト

# ウェブサービスの問題点(1)

- タイムアウト
  - ネット越しにアクセスするので不安定

```
# 長めに設定
serv.options["protocol.http.connect_timeout"] = 60
serv.options["protocol.http.receive_timeout"] = 600

# それでも接続が切れることはある
begin
  results = serv.send(*arg)
rescue Timeout::Error
  retry
end
```

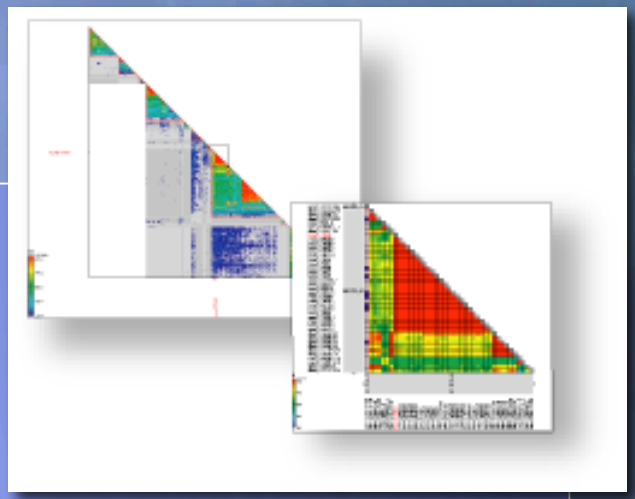
# ウェブサービスの問題点(2)

- Proxy
  - 環境変数http_proxyだけではダメな場合も

```
# Ruby (SOAP4R) の場合
setenv SOAP_USE_PROXY on
setenv HTTP_PROXY my.proxy.server:8080
```

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

use strict;
use SOAP::Lite;

my $wsdl = "http://soap.genome.ad.jp/KEGG.wsdl";
my $results = SOAP::Lite
                -> proxy("$wsdl", proxy => "http://my.proxy.server/")
                -> get_pathways_by_enzymes(
                        SOAP::Data->name(data=>['ec:1.3.99.1']));

foreach (@{$results}) { print $_,"\n"; }
```

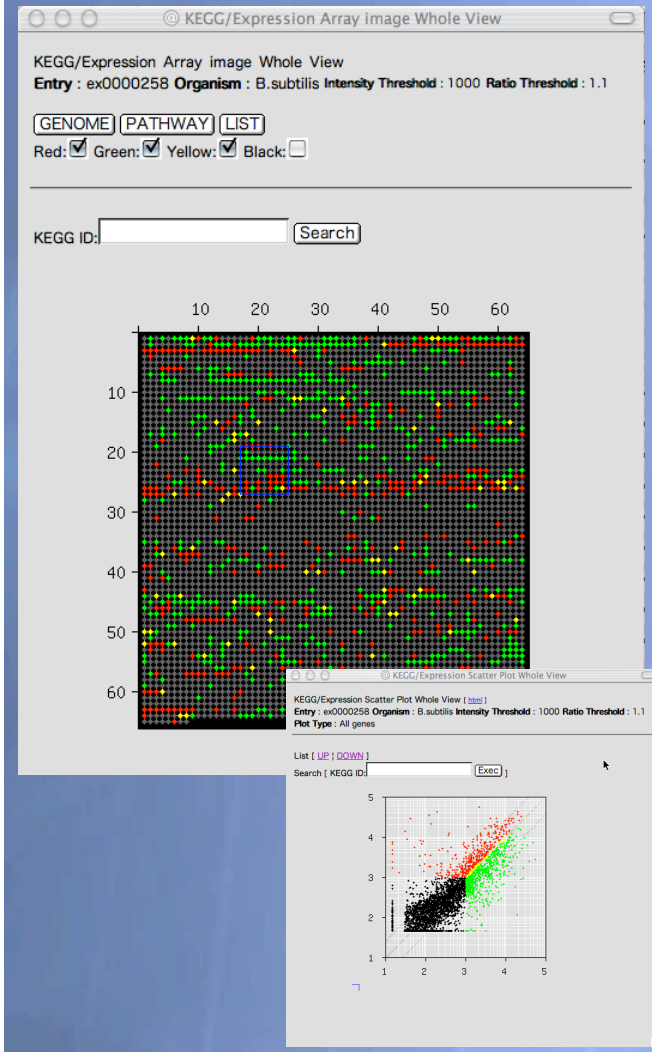
# 例：ホモログ遺伝子のモチーフ検索



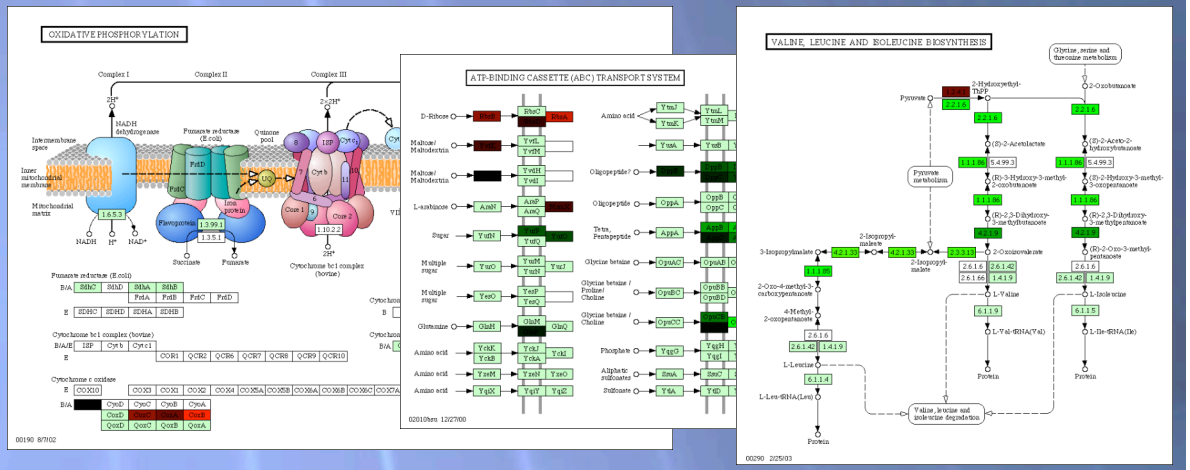
```ruby
#!/usr/bin/env ruby

require 'bio'

serv = Bio::KEGG::API.new

# ヒト遺伝子 hsa:7368 の他生物種でのホモログを検索
homologs = serv.get_all_best_neighbors_by_gene("hsa:7368")

homologs.each do |hit|
  gene = hit.genes_id2
  # 各ホモログ遺伝子についてモチーフを検索、あれば遺伝子名と共に表示
  if motifs = serv.get_motifs_by_gene(gene, "pfam")
    motifs.each do |motif|
      name = motif.motif_id
      desc = motif.definition
      puts "#{gene}: #{name} #{desc}"
    end
  end
end
```

# 例：遺伝子発現をパスウェイにマッピング

```
serv = Bio::KEGG::API.new
```

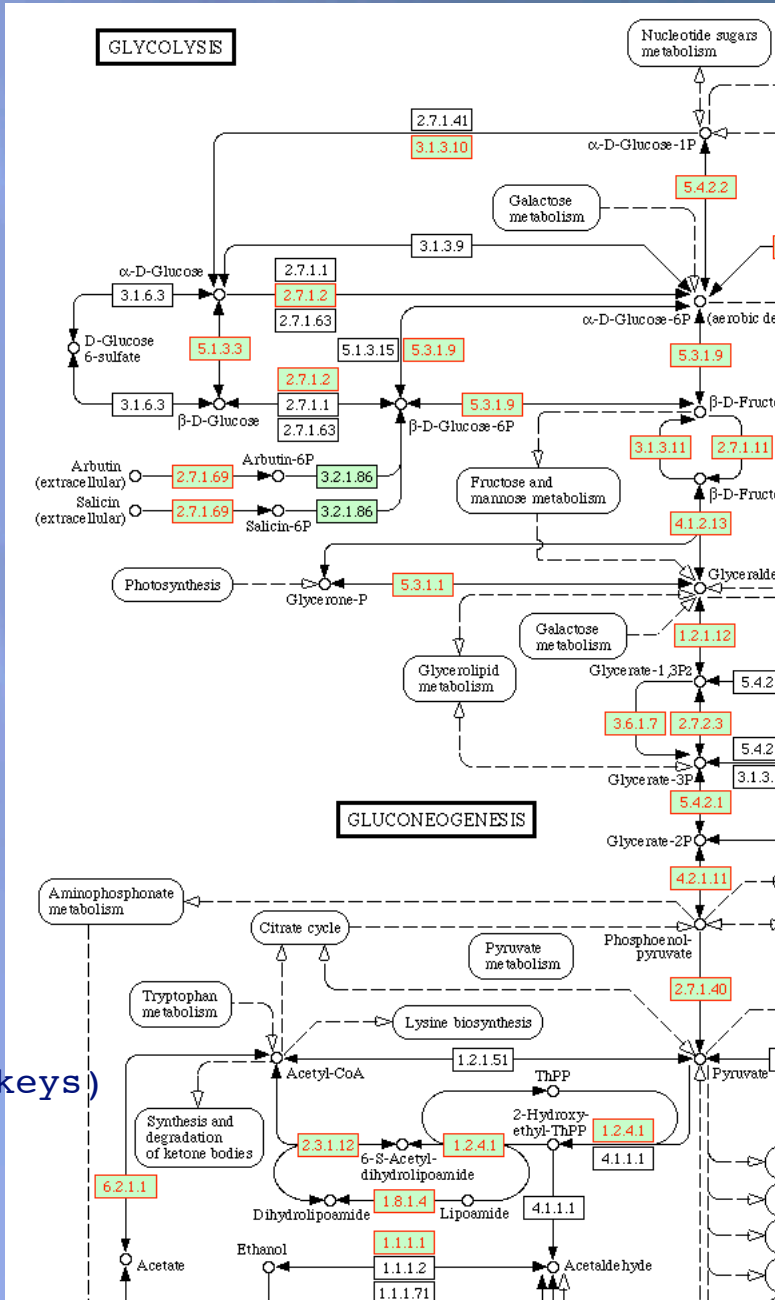



```
list = serv.get_genes_by_pathway("path:bsu00020")
fg_colors = Array.new
bg_colors = Array.new

list.each do |gene|
  fg_colors << "black"
  bg_colors << ratio2rgb(gene) # 遺伝子名と色の対応
end

url = serv.color_pathway_by_objects(
  "path:bsu00020", list, fg_colors, bg_colors)
```



# 例：PDB へのマッピング

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

require 'bio'

serv = Bio::KEGG::API.new

# 指定したいパスウェイ上の遺伝子のリスト
path = ARGV.shift || "path:eco00010"
genes = serv.get_genes_by_pathway(path)

# PDBにリンクのある遺伝子を検索
results = Hash.new
genes.each do |gene|
  if pdb_links =
      serv.get_all_linkdb_by_entry(gene, "pdb")
    pdb_links.each do |link|
      results[gene] = true
    end
  end
end

# 色付き画像を生成
url = serv.mark_pathway_by_objects(path, results.keys)

# 画像を保存
serv.save_image(url, "pdb.gif")
```



# 例：相互運用
# マルチプルアライメント

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

require 'bio'

### KEGG API

kegg = Bio::KEGG::API.new

list = kegg.get_all_paralogs_by_gene("eco:b0002")
genes = Array.new
list.each do |hit|
  genes << hit.genes_id2
end
seqs = kegg.get_aaseqs(genes)

### DDBJ XML

ddbj = Bio::DDBJ::XML::ClustalW.new

puts ddbj.analyzeSimple(seqs)
```

実行例はターミナルにて

# 相互運用における課題

- 同じ遺伝子を指すのにデータベース間でIDが共通でない
  - たとえば DDBJ で検索して得た遺伝子がKEGG のパスウェイのどこに載るのか
  - LSID？
  - データベース間のオブジェクトを対応づけるウェブサービス？

# ウェブサービス統合の試み

- BioMOBY
  - バイオインフォの様々なウェブサービス二対する UDDI 的なのディレクトリサービス（のはず）
  - http://www.biomoby.org/

# ウェブサービスの勧め

- データのモデル化が改善
- アプリケーション例の拡大
- ユーザ層、アクセス数の拡大
- ユーザはHTMLやXMLのパース地獄から脱却

- もっと多様なサービスが増えて欲しい

# KEGG API開発の経緯

- KEGG の機能をクリックせずに使いたい
  - 大量にバッチ処理したい
  - ウェブページの HTML をパースしたくない
- Oracle に格納されているデータを柔軟に取り出したいが SQL を直接ユーザに書かせたくはない
  - HTML/CGI のフォームではニーズへの対応に限界
- Perl の SOAP::Lite でプロトタイプ作成
  - わりと簡単だった＆他の言語からも利用できた

# KEGG APIにできること

- PATHWAY
  - 載っている遺伝子、化合物などの検索
  - 画像に対する自由な色づけ
- GENES/SSDB
  - ゲノムの決まった全遺伝子のカタログ
  - pre-calc な遺伝子間の類似度データベース
  - オーソログ、パラログの推定
  - Pfam などのモチーフ情報
- DBGET
  - ゲノムネットの全データベース検索エントリ取得

# KEGG API導入による効果

- ゲノムネットの頻繁に変わる HTML をパーズしなくて良くなった
- RDB への安全で柔軟なアクセス
- KEGG を使ったユーザの解析の自由度
- アクセス数の向上
- アプリケーションからの利用
- ユーザからの新たな応用例