JSTゲノムリテラシー講座
ゲノムアノテーション -データベースとツールの活用-

KEGG DASとKEGG APIで繋がる ゲノムアノテーションとパスウェイ

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>
http://bioruby.org/
http://www.genome.jp/kegg/soap/
http://das.hgc.jp/

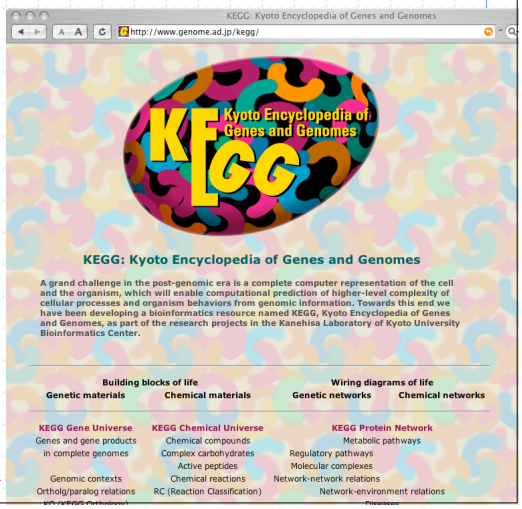
2004/08/24@サイエンスプラザ

目次

- KEGG (Kyoto Encyclopedia of Genes and Genomes)
  - GENES
  - KO (KEGG Orthology)
  - SSDB
  - OC (Ortholog Cluster)
  - PATHWAY
- KEGG DAS
  - DAS (Distributed Annotation System)
- KEGG API
  - SOAP/WSDL
- BioRuby
  - Ruby (Object oriented scripting language)

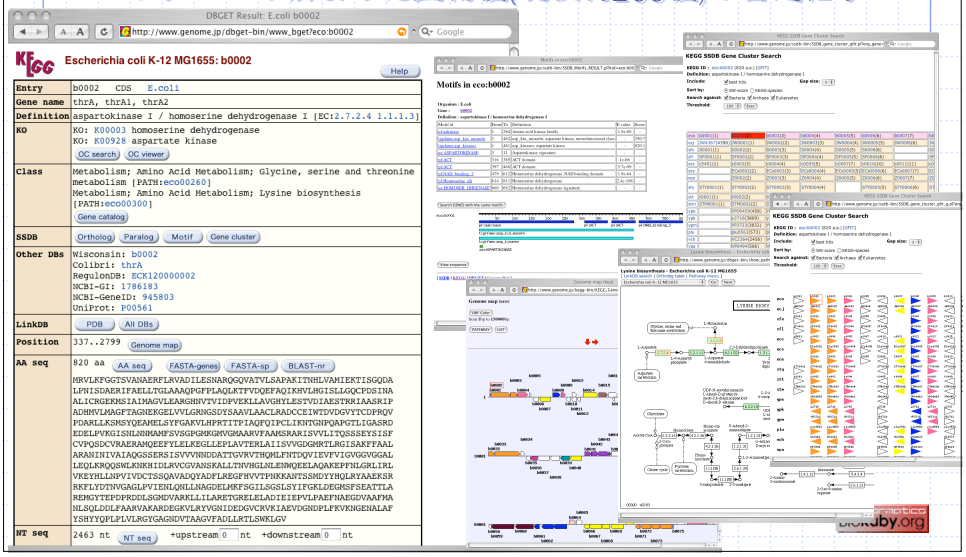
KEGG

- http://www.genome.jp/kegg/
- 京都大学化学研究所バイオインフォマティクスセンター(金久研)で開発されているデータベース
  - PATHWAY
    - LIGAND
  - GENES, GENOME
  - KO
  - SSDB
  - Expression
  - Glycan
  - etc.



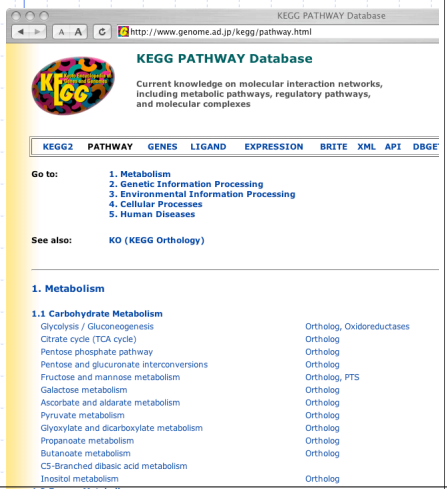
KEGGのゲノムアノテーション

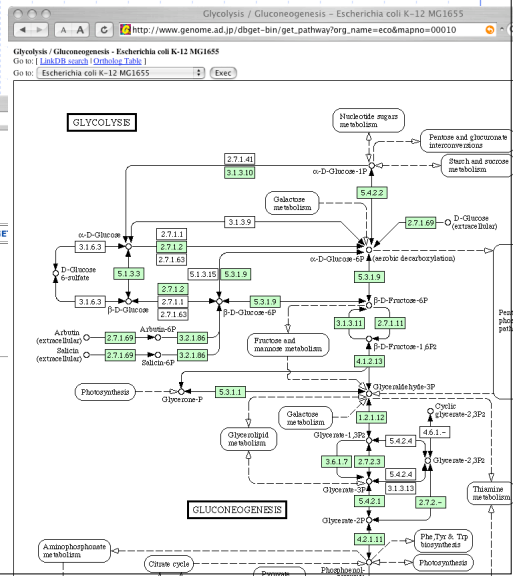
- GENES - protein universe
  - ゲノムの決まった生物種(現在約200種)の全遺伝子



KEGGのパスウェイ

- 遺伝子間のつながりをグラフで表現したもの
  - 代謝系
  - 制御系



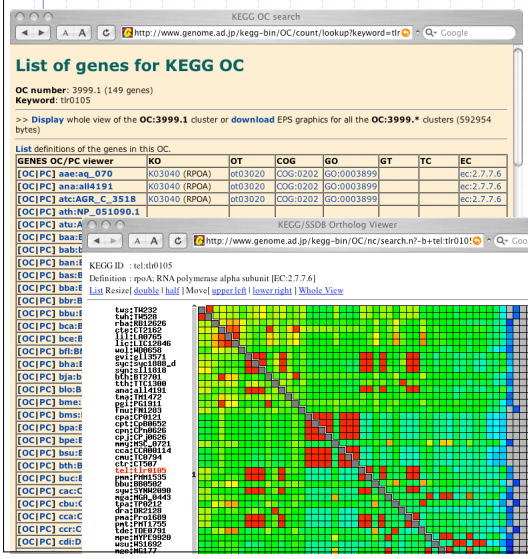


KO (KEGG Orthology)

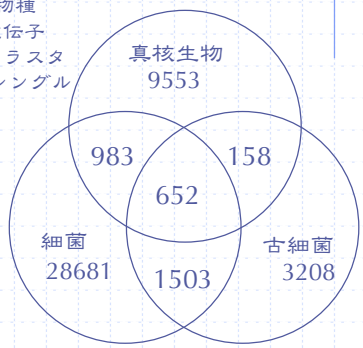
- パスウェイへのアノテーションを効果的に進めるため、オーソログ対応の取れる遺伝子をキュレーション
  - 酵素番号より細かい分類
  - パスウェイ上の各ボックスの分類に相当
  - 人手で行なうため時間がかかる

OCとSSDB

- 網羅的に候補を抽出するために全遺伝子ペアの配列類似性を計算しSSDBに格納、機械的にクラスタリング



173生物種
59万遺伝子
12万クラスタ
7.6万シングル



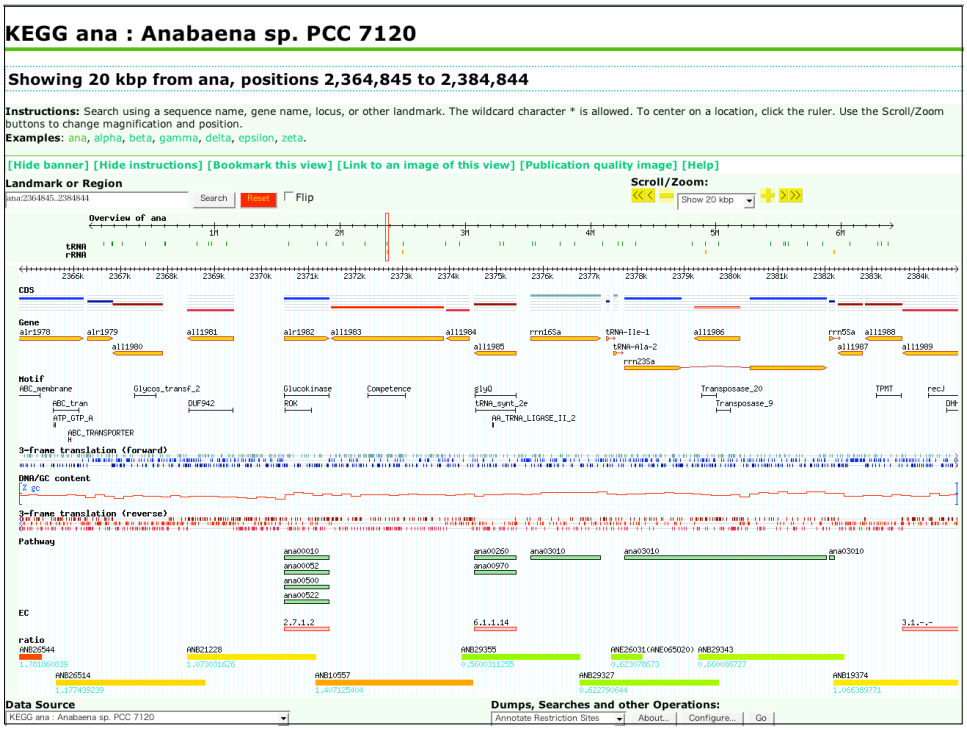
３生物種界でのOCの内訳

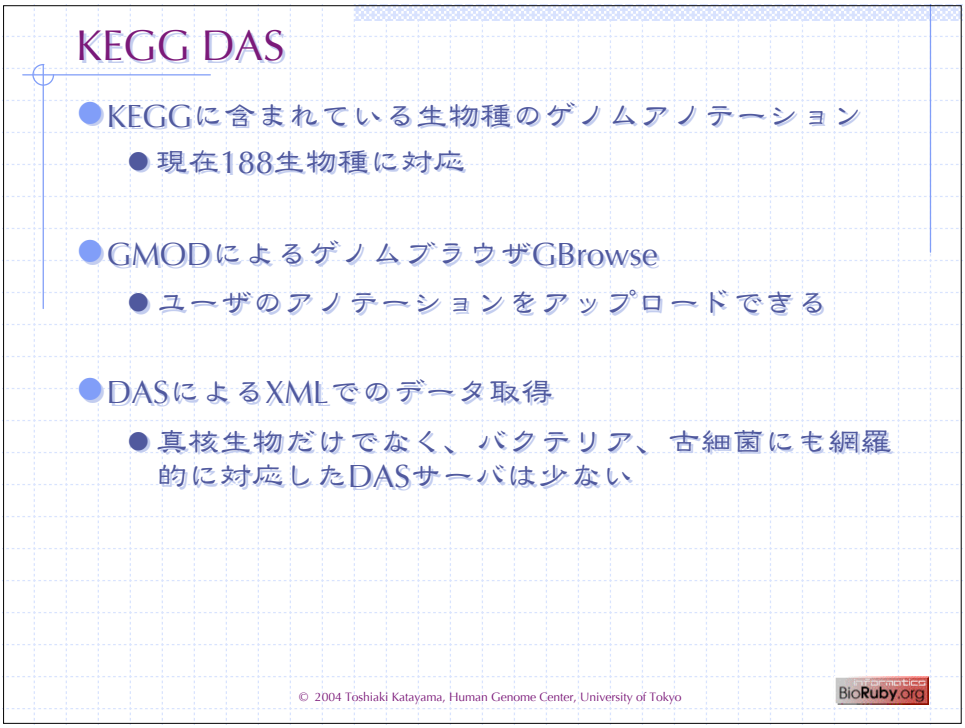
BioRuby.org

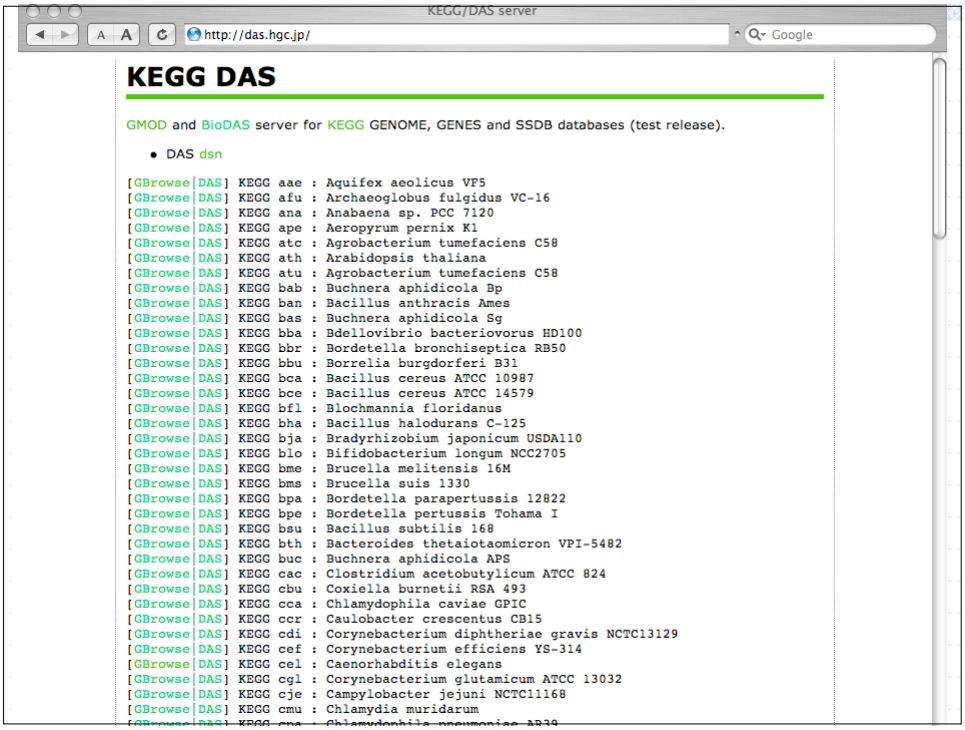
KEGGを見てみよう

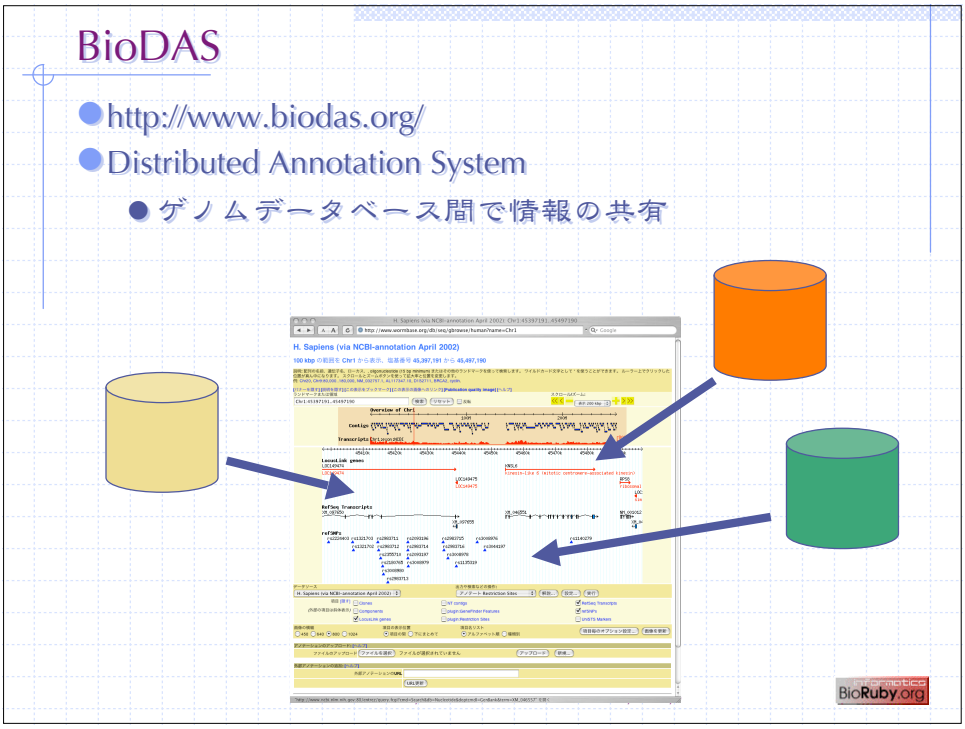
- http://www.genome.jp/
  - 遺伝子検索
  - GENESエントリ
  - LinkDB
  - パスウェイへのリンク
  - オーソログクラスター
  - 遺伝子クラスター
  - etc.

BioRuby.org


KEGG ana : Anabaena sp. PCC 7120
Showing 20 kbp from ana, positions 2,364,845 to 2,384,844
Instructions: Search using a sequence name, gene name, locus, or other landmark. The wildcard character * is allowed. To center on a location, click the ruler. Use the Scroll/Zoom buttons to change magnification and position.
Examples: ana, alpha, beta, gamma, delta, epsilon, zeta.
[Hide banner] [Hide instructions] [Bookmark this view] [Link to an image of this view] [Publication quality image] [Help]
Landmark or Region
Search
Reset
Flip
Scroll/Zoom:
Show 20 kbp
Overview of ana
tRNA
rRNA
CDS
Gene
Motif
3-frame translation (forward)
DNA/GC content
3-frame translation (reverse)
Pathway
EC
ratio
Data Source
KEGG ana : Anabaena sp. PCC 7120
Dumps, Searches and other Operations:
Annotate Restriction Sites
About...
Configure...
Go


KEGG DAS
KEGGに含まれている生物種のゲノムアノテーション
現在188生物種に対応
GMODによるゲノムブラウザGBrowse
ユーザのアノテーションをアップロードできる
DASによるXMLでのデータ取得
真核生物だけでなく、バクテリア、古細菌にも網羅的に対応したDASサーバは少ない
© 2004 Toshiaki Katayama, Human Genome Center, University of Tokyo
BioRuby.org


KEGG/DAS server
http://das.hgc.jp/
Google
KEGG DAS
GMOD and BioDAS server for KEGG GENOME, GENES and SSDB databases (test release).
DAS dsn
[GBrowse|DAS] KEGG aae : Aquifex aeolicus VF5
[GBrowse|DAS] KEGG afu : Archaeoglobus fulgidus VC-16
[GBrowse|DAS] KEGG ana : Anabaena sp. PCC 7120
[GBrowse|DAS] KEGG ape : Aeropyrum pernix K1
[GBrowse|DAS] KEGG atc : Agrobacterium tumefaciens C58
[GBrowse|DAS] KEGG ath : Arabidopsis thaliana
[GBrowse|DAS] KEGG atu : Agrobacterium tumefaciens C58
[GBrowse|DAS] KEGG bab : Buchnera aphidicola Bp
[GBrowse|DAS] KEGG ban : Bacillus anthracis Ames
[GBrowse|DAS] KEGG bas : Buchnera aphidicola Sg
[GBrowse|DAS] KEGG bba : Bdellovibrio bacteriovorus HD100
[GBrowse|DAS] KEGG bbr : Bordetella bronchiseptica RB50
[GBrowse|DAS] KEGG bbu : Borrelia burgdorferi B31
[GBrowse|DAS] KEGG bca : Bacillus cereus ATCC 10987
[GBrowse|DAS] KEGG bce : Bacillus cereus ATCC 14579
[GBrowse|DAS] KEGG bfl : Blochmannia floridanus
[GBrowse|DAS] KEGG bha : Bacillus halodurans C-125
[GBrowse|DAS] KEGG bja : Bradyrhizobium japonicum USDA110
[GBrowse|DAS] KEGG blo : Bifidobacterium longum NCC2705
[GBrowse|DAS] KEGG bme : Brucella melitensis 16M
[GBrowse|DAS] KEGG bms : Brucella suis 1330
[GBrowse|DAS] KEGG bpa : Bordetella parapertussis 12822
[GBrowse|DAS] KEGG bpe : Bordetella pertussis Tohama I
[GBrowse|DAS] KEGG bsu : Bacillus subtilis 168
[GBrowse|DAS] KEGG bth : Bacteroides thetaiotaomicron VPI-5482
[GBrowse|DAS] KEGG buc : Buchnera aphidicola APS
[GBrowse|DAS] KEGG cac : Clostridium acetobutylicum ATCC 824
[GBrowse|DAS] KEGG cbu : Coxiella burnetii RSA 493
[GBrowse|DAS] KEGG cca : Chlamydophila caviae GPIC
[GBrowse|DAS] KEGG ccr : Caulobacter crescentus CB15
[GBrowse|DAS] KEGG cdi : Corynebacterium diphtheriae gravis NCTC13129
[GBrowse|DAS] KEGG cef : Corynebacterium efficiens YS-314
[GBrowse|DAS] KEGG cel : Caenorhabditis elegans
[GBrowse|DAS] KEGG cgl : Corynebacterium glutamicum ATCC 13032
[GBrowse|DAS] KEGG cje : Campylobacter jejuni NCTC11168
[GBrowse|DAS] KEGG cmu : Chlamydia muridarum


BioDAS
http://www.biodas.org/
Distributed Annotation System
ゲノムデータベース間で情報の共有
H. Sapiens (via NCBI-annotation April 2002)
BioRuby.org

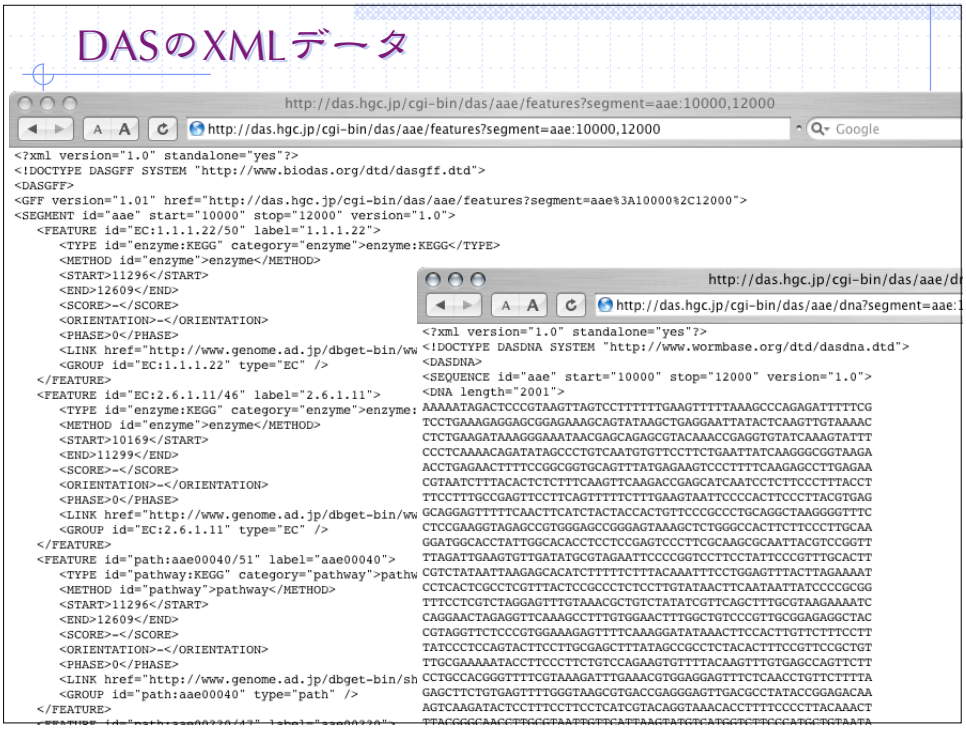
GMOD

- Generic Model Organism Database
  - WormBase, FlyBase, SGD などのモデル生物で共通に利用可能でオープンなゲノムデータベースを開発
    - データベーススキーマ Chado (茶道)
    - ゲノムブラウザ GBrowse
    - アノテーションツール Apollo

KEGG DASを使ってみる

- http://das.hgc.jp/
  - 生物種の選択
  - クロモソームの選択
  - スクロール、ズーム
  - KEGGのアノテーションへリンク
  - ユーザのアノテーションをアップロード

DASのXMLデータ



DASのURL

- 裏ではURLにより領域を指定、データ取得
  - XMLが返されるので必要な情報を抽出
- 塩基配列の取得
  - http://das.hgc.jp/cgi-bin/das/eco/dna?segment=eco:101,200
- アノテーション情報の取得
  - http://das.hgc.jp/cgi-bin/das/sce/features?segment=I:5001,9000
- BioRubyでデータ取得する方法は後ほど

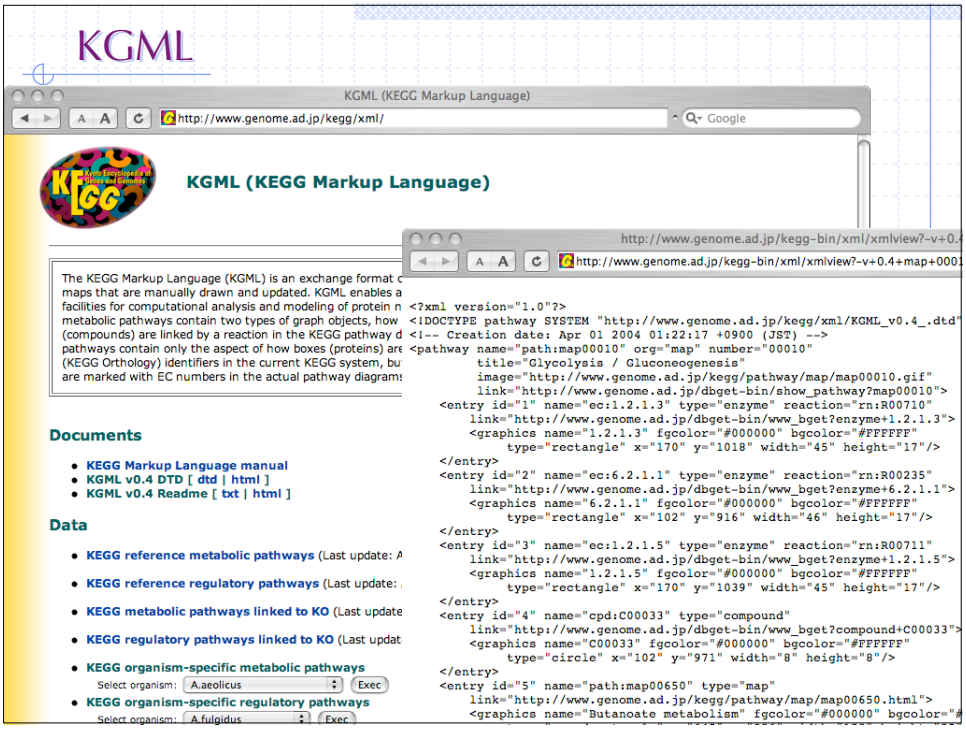
KEGGの標準化

- KEGG DAS
  - GBrowse/DASによるゲノムアノテーション
  - ゲノム上での遺伝子の情報
- KEGG API
  - SOAP/WSDLによるKEGGへのアクセス
  - パスウェイ上の遺伝子群や類似遺伝子群の情報
- KGML
  - XMLによるKEGG PATHWAYの表現
  - 生物種ごとに遺伝子や化合物間のつながりの情報

KEGG API

- パスウェイやアノテーション情報をプログラムで取得
- カテゴリ（マニュアル参照）
  - メタ情報
  - DBGET
  - LinkDB
  - SSDB
  - Motif
  - KO, OC, PC
  - PATHWAY - get_genes_by_pathway(pathway_id)
  - GENES
  - GENOME
    - 具体的なメソッドや実際例は後ほど

KGML



RubyとBioRuby

- Ruby とは
  - 国産でフリーのオブジェクト指向スクリプト言語
  - すっきりとした分かりやすい文法
    - 記号が少ない、便利なイテレータ(ブロック)
  - 充実した組み込みクラスや標準添付ライブラリ
    - 数値、文字列、配列、ハッシュ、正規表現、...
    - クラスごとに使えるメソッドが豊富
- BioRubyとは
  - Rubyのバイオインフォマティクス用ライブラリ
  - 2000年から主に日本で開発、KEGGなどもサポート

なぜRuby

- Rubyはどのような場面に向いているか
  - Perl同様、文字列処理や正規表現が得意
  - 遺伝子の配列やアノテーションなど文字情報が多い生物学のデータを扱うのに有利 → Perlの成功
- Ruby の代わりに Perl, Python でも全く問題はない
  - 普及率と使いやすさ（慣れ？）のトレードオフ
  - 使いたいライブラリがあるかどうか
  - JavaScript, PHP, SQL なども知っていると便利
- C, C++, Java などはコンパイルが必要
  - プログラムが長くなりがち→手軽ではない
  - 高速な処理が可能、GUI アプリなどにも向く

Perlは？

- スクリプト言語の草分けでたぶん最も普及している
  - ユーザ数が多く、ライブラリも豊富
  - BioPerlがある
- でもデータ構造に弱い
  - 配列とハッシュの組み合わせ
  - Perl5以降のオブジェクト指向機能でカバー
    - 記号やオマジナイの増加 (リファレンス, bless, @ISA, ...)
- さらに基本的に使い捨てプログラム用の言語なので
  - 他人のPerlスクリプトは読めないことが多い
    - しばらく前に自分が書いたプログラムでも、、

Open Bio* とは

- PerlにはBioPerl, PythonにはBiopython, JavaにはBioJava
  言語ごとにライブラリ作成を進めているコミュニティが
- Open Bio Foundation
  - これら全体を支援し、BOSC, BioHackathonを開催

BioRuby.org

BOSC

- ISMBと併催されるOpen Bio*のミーティング(BOF)
  - 進捗など成果発表を行ない、コミュニケーション

BioHackathon

- BOSCと異なりクローズド（企業スポンサー）
  - ホテルに缶詰になりハックしまくる
- 第１回 2002/01 アリゾナ
  - O'Reilly
- 第２回 2002/02 南アフリカ
  - Electric Genetics
- 第３回 2003/02 シンガポール
  - Apple Singapore

BioRuby.org

Schedule – hack hack hack

BIO HACKATHON TIME MANAGEMENT PLAN DATE: 24 FEB 02 - 02 MAR 02

Starting	Ending	SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		24-Feb-02	25-Feb-02	26-Feb-02	27-Feb-02	28-Feb-02	1-Mar-02	2-Mar-02
7:00 AM	7:30 AM							
7:30 AM	8:00 AM							
8:00 AM	8:30 AM							
8:30 AM	9:00 AM		Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	
9:00 AM	9:30 AM							
9:30 AM	10:00 AM							
10:00 AM	10:30 AM							
10:30 AM	11:00 AM		Coffee Break	Coffee Break	Coffee Break	Coffee Break	Coffee Break	
11:00 AM	11:30 AM		Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	
11:30 AM	12:00 PM							
12:00 PM	12:30 PM							
12:30 PM	1:00 PM		Lunch	Lunch	Lunch	Lunch	Lunch	
1:00 PM	1:30 PM							
1:30 PM	2:30 PM		Hack Hack Hack	Off site Touring	Hack Hack Hack	Hack Hack Hack	Hack Hack Hack	
2:00 PM	2:30 PM			Dive / Snorkelling				
2:30 PM	3:00 PM			Kirstenbosch hike				
3:00 PM	3:30 PM			Waterfront				
3:30 PM	4:00 PM		Coffee Break		Coffee Break			
4:00 PM	4:30 PM	Opening Reception &	Hack Hack Hack		Hack Hack Hack	Township Tour &	Coffee Break	
4:30 PM	5:00 PM	Cocktails & Snacks				African Café	Closing remarks	
5:00 PM	5:30 PM			Hack Hack Hack				
5:30 PM	6:00 PM							
6:00 PM	6:30 PM	Table Mountain						
6:30 PM	7:00 PM							
7:00 PM	7:30 PM		Dinner outside	Dinner at Hotel	Dinner at Hotel		Closing Party Outside	
7:30 PM	8:00 PM		at Castle					
8:00 PM	8:30 PM							
8:30 PM	9:00 PM			Hack Hack Hack	Hack Hack Hack	Hack Hack Hack		
9:00 PM	9:30 PM							
9:30 PM	10:00 PM							
10:00 PM	10:30 PM							
10:30 PM	11:00 PM							
11:00 PM	11:30 PM							
11:30 PM	12:00 AM							
12:00 AM	12:30 AM							
12:30 AM	1:00 AM							

BioRuby.org

Hacking Room

ラボ訪問やJunior BioHackathonも

- A*starの研究所のメンバーとディスカッション
- 14-16才くらいの子供たちへのレクチャー

スポンサー募集中...

BioRubyのカバーする領域

- Bio::Sequence, Bio::Location, Bio::Feature
  - 配列操作、スプライシング、ウィンドウサーチなど
- Bio::Registry, Bio::SQL, Bio::Fetch, Bio::FlatFile
  - OBDAによるデータベースへのアクセス
- Bio::DB
  - GenBank, SwissProt, KEGG などデータベースのパーザ
- Bio::Blast, Bio::Fasta (他 HMMER, MAFFT, Sosui, Psort etc.)
  - 解析アプリケーションの実行と結果のパーザ
- Bio::DAS, Bio::KEGG::API, Bio::DDBJ::XML
  - ウェブサービス
- Bio::Pathway, Bio::Relation
  - グラフ計算
- Bio::PubMed, Bio::Reference
  - 文献情報

配列操作

- スプライシング
- 相補鎖
- 翻訳
- 組成
- 分子量
- windowサーチ

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

require 'bio'

seq = Bio::Sequence::NA.new(ARGF.read)

puts seq.to_fasta("foo", 60)
puts seq.subseq(1,3)
puts seq.splicing("join(1..23,45..67)")
puts seq.complement
puts seq.translate
puts seq.gc_percent
puts seq.composition

seq.window_search(15, 3) do |subseq|
  peptide = subseq.translate
  puts peptide.molecular_weight
end
```


データベースへのアクセス

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

require 'bio'

serv  = Bio::Registry.new
db    = serv.get_database("swissprot")
entry = db.get_by_id("TETW_BUTFI")
puts entry
```

OBDA

KEGG API

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

require 'bio'

serv  = Bio::KEGG::API.new
entry = serv.bget("sp:TETW_BUTFI")
puts entry
```

OBDA (Open Bio* Database Access)

- BioHackathon (2002, 2003)で制定
  - データベースのエントリを取得する仕組み
    - BioPerl, BioPython, BioJava, BioRubyで共通化
- 設定ファイル
  - ~/.bioinformatics/seqdatabase.ini
  - /etc/bioinformatics/seqdatabase.ini
  - http://open-bio.org/registry/seqdatabase.ini
  - チュートリアルを参照

OBDA configuration

- ~/.bioinformatics/seqdatabase.ini
- /etc/bioinformatics/seqdatabase.ini
- http://www.open-bio.org/registry/seqdatabase.ini

```
VERSION=1.00

[genbank]
protcol=flat
location=/export/database/
dbname=genbank

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

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

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

require 'bio'

reg = Bio::Registry.new

sp = reg.get_database('swissprot')
puts sp.get_by_id('CYC_BOVIN')

gb = reg.get_database('genbank')
puts gb.get_by_id('AA2CG')
```

データベースエントリのパーサ

- GenBank, GenPept, RefSeq, DDBJ
- EMBL, UniProt (TrEMBL, SwissProt)
- PDB
- KEGG/GenomeNet
  - GENES, GENOME, ENZYME, COMPOUND, KO, BRITE, CELL, Expression, Keggtab, AAindex
- GFF
- GO
- FANTOM
- Transfac, Prosite
- LITDB, MEDLINE
- NBRF, PIR
- FASTA format

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

require 'bio'

Bio::FlatFile.auto(ARGF) do |ff|
  ff.each do |entry|
    # do something
  end
end
```


BioRuby.org

解析ソフトの実行

- Bio::Blast, Fasta, HMMER, EMBOSS
- Bio::ClustalW, MAFFT
- Bio::Genscan
- Bio::PSORT, TargetP
- Bio::SOSUI, TMHMM

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

File.open("my_blast_output.xml") do |file|
  Bio::Blast.reports(file) do |report|
    report.hits do |hit|
      hit.each do |hsp|
        puts hsp.query_id, hsp.target_id,
        hsp.bit_score, hsp.evalue, hsp.overlap
    end
  end
end
```

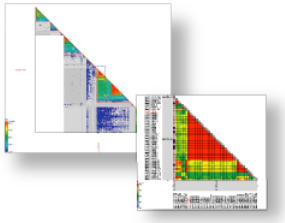
BioRuby.org

サンプル集

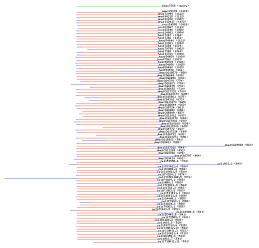
- KEGG APIとKEGG DASをBioRubyから使ってみる
  - SSDBを使ってホモログとモチーフを検索
  - KO, OC, PC で仲間の遺伝子を検索
  - データベースの情報やエントリの取得
  - パスウェイと遺伝子の対応
  - パスウェイに色をつける

BioRuby.org

Example1：SSDBを使ってホモログとモチーフを検索



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

require 'bio'

# KEGG APIのサーバに接続
serv = Bio::KEGG::API.new

# 各生物種のbest-hit遺伝子を取得
homologs = serv.get_all_best_neighbors_by_gene("hsa:7368")

homologs.each do |hit|
  # hitした遺伝子名を取り出す
  gene = hit.genes_id2
  # KEGG APIで検索して遺伝子がモチーフを持っていれば
  if motifs = serv.get_motifs_by_gene(gene, "pfam")
    motifs.each do |motif|
      # 各モチーフのIDと説明を取り出してタブ区切りで表示
      name = motif.motif_id
      desc = motif.definition
      puts "#{gene}:¥t#{name}¥t#{desc}"
    end
  end
end
```



BioRuby.org

Example2: KO, OC, PCで仲間の遺伝子

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

require 'bio'

serv = Bio::KEGG::API.new

# 遺伝子のKO番号のリストを取得
list = serv.get_ko_by_gene("eco:b0002")
# 例として１つ目のKO番号を使う
ko_id = list.first

# 同じKO (KEGG Orthology) がアサインされた遺伝子のリスト
ko_genes = serv.get_ko_members(ko_id)

# 同じOC (Ortholog Cluster) がアサインされた遺伝子のリスト
oc_genes = serv.get_all_oc_members_by_gene("hsa:7368")

# 同じPC (Paralog Cluster) がアサインされた遺伝子のリスト
pc_genes = serv.get_all_pc_members_by_gene("hsa:7368")

puts "# KO", ko_genes
puts "# OC", oc_genes
puts "# PC", pc_genes
```


Example3: データベース情報やエントリ取得

```
#!/usr/bin/ruby

require 'bio'

serv = Bio::KEGG::API.new

# KEGGに現在含まれている生物種のリスト
orgs = serv.list_organisms
puts "# list of organisms in KEGG"
orgs.each do |entry|
  puts "#{entry.entry_id} #{entry.definition}"
end

# KEGGで利用可能なヒト(hsa)のパスウェイのリスト
list = serv.list_pathways("hsa")
puts "# list of pathways for human in KEGG"
list.each do |entry|
  puts "#{entry.entry_id} #{entry.definition}"
end

# 複数のKEGG GENESエントリを取得
puts "Human gene entries for gene 7368 and 7369"
puts serv.bget("hsa:7368 hsa:7369")
```

Example4: PATHWAYと遺伝子の対応

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

require 'bio'

serv = Bio::KEGG::API.new

# パスウェイ上の遺伝子のリスト
genes = serv.get_genes_by_pathway("path:hsa00020")
puts "# genes on human's pathway 00020"
genes.each do |gene|
  puts gene
end

# EC番号を遺伝子名に対応づけ
list = ["ec:1.1.1.1", "ec:1.2.1.1"]
list.each do |ec|
  puts "# E. coli genes for #{ec}:"
  puts serv.get_genes_by_enzyme(ec, "eco")
end
```


BioRuby.org

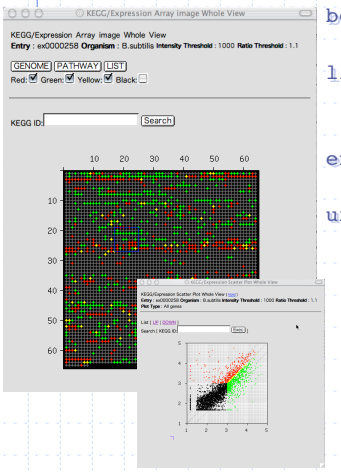
Example5: PATHWAYに色を付ける

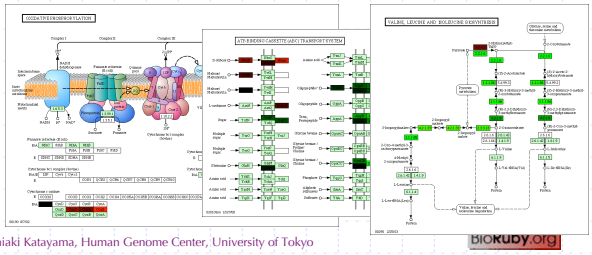
```
#!/usr/bin/env ruby

require 'bio'

serv = Bio::KEGG::API.new

# パスウェイに赤枠でマークをつける
objs = ['eco:b0002', 'cpd:C00263']
url1 = serv.mark_pathway_by_objects('path:eco00260', objs)
puts "# URL for marked pathway:", url1

# パスウェイに色をつける
fg_list = ['blue', 'green']
bg_list = ['#ff0000', 'yellow']
url2 = serv.color_pathway_by_objects('path:eco00260',
         objs, fg_list, bg_list)
puts "# URL for colored pathway:", url2

# 結果の画像を保存する
serv.save_image(url1, "example5-1.gif")
serv.save_image(url2, "example5-2.gif")
```


BioRuby.org

Example6: 遺伝子発現をパスウェイにマッピング

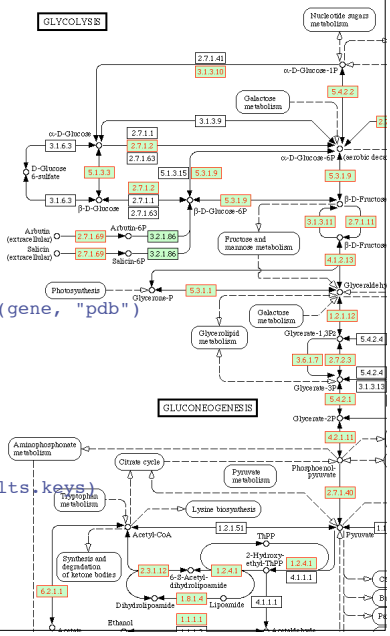
```
serv = Bio::KEGG::API.new

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)
```





Example7: 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, "example7.gif")
```



Example8: DASからの情報取得

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

require 'bio'

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

# 酵母のI番染色体1001塩基から2000塩基の範囲を指定、DNA配列を取得
segment = Bio::DAS::SEGMENT.region("I", 1001, 2000)
list = serv.get_dna("sce", segment)
list.each do |dna|
  puts dna.sequence
end

# 上記と同じ範囲で遺伝子アノテーションを取得
list = serv.get_features("sce", segment)
list.segments.each do |segment|
  segment.features.each do |feature|
    puts feature.entry_id
    puts feature.start
  end
end
```


BioRuby.org

なぜオブジェクト指向か

- そんなに大げさなものではなく、単に便利だから
  - BioPerl, Biopython, BioJava, BioRuby全て…
- オブジェクトはデータのいれもの（構造体）
  - さらにデータの操作方法（メソッド）を定義
    - 化合物オブジェクトには名前、構造式、機能説明などのデータ（情報）が入っている
    - 組成や分子量を計算するメソッドなど
- オブジェクトに共通の特徴を括り出したものがクラス
  - 抽象化（モノをどう捉えるか、理解するか）
  - 複数のクラスに共通の特徴を括り出したらスーパークラス（これを各クラスで継承し差分を書く）

BioRuby.org

クラスを作ってみる - Ruby入門

- いれものであるCpdクラスを宣言
  - 名前と構造式と説明を覚えておく

```
class Cpd

  attr_accessor :name, :definition, :formula

end
```

```
a = Cpd.new
a.name = "ATP"
a.definition = "Adenosine 5'-triphosphate"
a.formula = "C10H16N5O13P3"

puts a.name       # => "ATP"
puts a.definition # => "Adenosine 5'-tri.."
puts a.formula    # => "C10H16N5O13P3"
```


クラスを作ってみる - Ruby入門

- 本当なら...

```
class Cpd

  def initialize
    @formula = nil
  end

  def formula=(val)
    @formula = val
  end

  def formula
    return @formula
  end

end
```

```
cpd = Cpd.new
cpd.formula = "C10H16N5O13P3"
puts cpd.formula
# => "C10H16N5O13P3"
```

クラスを作ってみる - メソッド定義

- 構造式から組成を抽出するメソッドを追加

```
class Cpd

  attr_accessor :name, :definition, :formula

  # 組成を覚えておくHash型の変数を準備
  def initialize
    @comp = Hash.new
  end

  # 組成を抽出するメソッド "C10H16N5O13P3"
  def composition
    @formula.scan(/([A-Z]+)(\d+)/) do |a, b|
      @comp[a] = b.to_i
    end
    return @comp
  end

end
```

```
{ "C" => 10, "H" => 16,
  "N" => 5,  "O" => 13,
  "P" => 3 }
```


BioRuby.org

クラスを作ってみる - もう少し機能追加

- 組成から全体の分子量を計算するメソッドを追加

```
class Cpd
  # 手抜きのために分子量の表も内蔵
  MW = { 'C'=>12.011, 'H'=>1.00794, 'N'=>14.00674,
         'O' => 15.9994, 'P' => 30.973762 }

  def initialize
    @comp = Hash.new
  end
  attr_accessor :name, :definition, :formula

  def composition
    @formula.scan(/([A-Z]+)(\d+)/) do |a, b|
      @comp[a] = b.to_i
    end
    return @comp
  end

  def mol_weight
    total = 0.0
    composition.each do |elem, i|
      total += MW[elem] * i
    end
    return total
  end
end
```

BioRuby.org

クラスを作ってみる - 実行

- 以下の部分をつけてcpdtest1.rbに保存し、実行

```
# (続き)
cpd = Cpd.new
cpd.name = "ATP"
cpd.definition = "Adenosine 5'-triphosphate"
cpd.formula = "C10H16N5O13P3"

puts cpd.name
puts cpd.definition
puts cpd.formula
puts cpd.mol_weight
```

```
% ruby cpdtest1.rb
ATP
Adenosine 5'-triphosphate
C10H16N5O13P3
507.184226
```


クラスを作ってみる - ライブラリ化

- Cpdクラスの定義部分だけをcpd.rbに括り出し再利用

```
require 'cpd'

cpd = Cpd.new
cpd.name = "ATP"
cpd.definition = "Adenosine 5'-triphosphate"
cpd.formula = "C10H16N5O13P3"

puts cpd.name
puts cpd.definition
puts cpd.formula
puts cpd.mol_weight
```

```
% ruby cpdtest2.rb
ATP
Adenosine 5'-triphosphate
C10H16N5O13P3
507.184226
```

Acknowledgements

- BioRuby developers
  - Naohisa Goto, Mitsuteru Nakao, Yoshinori Okuji, Shuichi Kawashima, Masumi Itoh, Alex Gutteridge and some other contributors on the net.
- KEGG curators and KEGG API developers
  - Bioinformatics center, Human genome center
  - Yoko Sato, Miho Matsubayashi, Satoshi Miyazaki
- KEGG DAS
  - Mayumi Takashio, Mari Watanabe
- Open Bio* community