

Automatizzazione di ricerca biotecnologica

Davide Quaglia, Federico De Meo, Valerio Guarnieri

In questa esercitazione vedremo come realizzare dei programmi Java che si interfaccino al server KEGG allo scopo di aumentare l'automatizzazione nella ricerca biotecnologica.

Per fare ciò KEGG mette a disposizione delle API che, tramite protocollo SOAP, forniscono tutta una serie di metodi per recuperare le informazioni che si trovano sul database.

Iniziamo quindi a preparare l'ambiente di lavoro che fruttano queste API.

NOTA: in laboratorio si consiglia di operare nella cartella /tmp/ in cui vi è tutto lo spazio necessario; tale cartella viene svuotata ad ogni avvio della macchina per cui sui propri PC personali se ne sconsiglia l'uso.

Occorre creare una directory di lavoro (ad esempio di seguito chiamata KEGG/) all'interno della quale andremo a mettere i seguenti files:

- KEGG.wsdl (file che descrive i web services forniti da KEGG)
- keggapi.jar (le api fornite da KEGG)

trovate tutti questi files sulla pagina di KEGG riservata a SOAP: <http://www.genome.jp/kegg/soap/>. E' anche disponibile il manuale Javadoc (non molto dettagliato) della libreria che abbiamo scaricato, reperibile al link: http://www.genome.jp/kegg/soap/doc/keggapi_javadoc/.

Prima di procedere, è necessario anche scompattare l'archivio contenente java axis dal seguente link: http://www.apache.org/dyn/closer.cgi/ws/axis/1_4/

Selezionate uno dei mirror presenti sulla pagina (sotto la voce HTTP) e successivamente scegliete l'archivio axis-bin-1_4.tar.gz (ad esempio http://apache.fastbull.org/ws/axis/1_4/axis-bin-1_4.tar.gz).

Scompattate l'archivio appena scaricato e copiate tutti i file .jar dalla cartella lib/ alla vostra cartella di lavoro KEGG/.

Infine occorre mettere nella cartella di lavoro tutti i programmi Java che si vogliono compilare ed eseguire.

Al momento della compilazione e della esecuzione dei nostri programmi Java dovremmo aver cura di comprendere i file appena scaricati, per farlo dobbiamo includere nel classpath le librerie scaricate come spiegato di seguito.

Portarsi nella cartella KEGG/

Compilazione:

```
javac -classpath ../axis.jar:../jaxrpc.jar:../wsdl4j-1.5.1.jar:../keggapi.jar *.java
```

Esecuzione:

```
java -classpath ../axis.jar:../jaxrpc.jar:../wsdl4j-1.5.1.jar:../keggapi.jar:../commons-logging-1.0.4.jar:../commons-discovery-0.2.jar:../saaj.jar classe [parametri]
```

NOTA: nel caso in cui la rete su cui si lavora richieda di accedere a Web mediante proxy (si evince dalla configurazione del Web browser o dalla configurazione di sistema) occorre specificare indirizzo e porta del proxy affinché le richieste HTTP escano della rete. Occorre modificare il comando di esecuzione come segue:

```
java -classpath ../axis.jar:../jaxrpc.jar:../wsdl4j-1.5.1.jar:../keggapi.jar:../commons-logging-1.0.4.jar:../commons-discovery-0.2.jar:../saaj.jar -Dhttp.proxyHost=proxy.univr.it -Dhttp.proxyPort=3128 classe [parametri]
```

Vediamo quindi subito un esempio:

```
import keggapi.*;

public class KeggOrganismsList {

    public static void main(String[] args) throws Exception {
        // creazione di un oggetti KEGGLocator, che ci "localizza" il server
        // KEGG
        KEGGLocator locator = new KEGGLocator();
        // con il metodo getKEGGPort stabiliamo la connessione a KEGG
        KEGGPortType serv = locator.getKEGGPort();
        // richiediamo la lista degli organismi presenti in KEGG
        Definition[] def = serv.list_organisms();
        // tramite un ciclo for stampiamo la lista degli organismi
        for (int i = 0; i < def.length; i++) {
            System.out.println(i+" "+def[i].getDefinition());
        }
    }
}
```

Compile ed eseguite il codice, come si evince dai commenti; questo semplice programma recupera la lista degli organismi contenuti nel database KEGG e li stampa a video.

Definition è una classe che contiene informazioni su un organismo; di conseguenza un array di Definition contiene un organismo per ogni valore.

Trovate la lista dei metodi della classe Definition sulla corrispondente pagina Javadoc:

http://www.genome.jp/kegg/soap/doc/keggapi_javadoc/keggapi/Definition.html

Esercizio:

Supponete ora di voler verificare la presenza di un certo organismo all'interno del database e di volerne stampare anche l'Entry_id oltre che al nome (vedi metodo corrispondente), modificate quindi il codice in modo da ricercare un particolare organismo passato come input sulla riga di comando. Suggerimento: ricavare la stringa con def[i].getDefinition() e su questa chiamare il metodo indexOf(args[0]).

Si noti che passando sulla linea di comando un nome di organismo senza delimitarlo da doppi apici solo la prima stringa viene messa in args[0] e quindi possono essere associati più risultati che contengono tale stringa .

Vediamo ora un altro esempio:

```
import keggapi.*;

public class GetGenesByEnzyme {
    public static void main(String[] args) throws Exception {
        // come sopra
        KEGGLocator locator = new KEGGLocator();
        KEGGPortType serv = locator.getKEGGPort();
        String results[] ;
        // recuperiamo il numero di geni dato l'organismo
        results = serv.get_genes_by_enzyme("ec:2.7.1.6", "ljo");
        System.out.println(results[0]);
    }
}
```

Il metodo `get_genes_by_enzyme()`, accetta come parametro due stringhe, una corrispondente all'enzima e l'altra corrispondente all'id dell'organismo.

I due sorgenti appena visti non avrebbero molto senso se eseguiti da soli, ma componendo il codice del primo con il codice del secondo si incomincia a vedere l'utilità di applicazioni che automatizzano la ricerca.

Esercizio:

Provate quindi a scrivere un programma che dato il nome di un organismo ne ricavi il suo id e dall'id ricavi il gene associato all'enzima `ec:2.7.1.6` (la spiegazione della scrittura `ec:2.7.1.6` si trova più avanti nella dispensa).

Provatelo con input: "Lactobacillus johnsonii NCC 533", il vostro codice dovrebbe restituire:

`ljo:LJ0859` (dove `ljo` è l'organismo e `LJ0859` il gene, questa scrittura sarà spiegata più avanti nella dispensa). Si noti che passando sulla linea di comando un nome di organismo senza delimitarlo da doppi apici solo la prima stringa viene messa in `args[0]` e quindi possono essere associati più risultati che contengono tale stringa. Questo caso deve essere rilevato (quanti parametri vedo sulla linea di comando?) e bisogna avvisare l'utente di delimitare il nome dell'organismo con doppi apici.

Vediamo ora alcuni esempi che restituiscono informazioni più complesse utilizzando particolari strutture dati caratteristiche di KEGG.

Iniziamo col vedere `SSDBRelation`:

```
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);
        }
    }
}
```

Questo codice è molto simile a quelli già visti, **dato un gene** ne restituisce i migliori vicini con relativo punteggio, provatelo con input “eco:b0002”.

Focalizziamo quindi l'attenzione su SSDBRelation.

Si tratta di una classe che contiene una serie di attributi di una entry del database, di seguito una lista dei dati di SSDBRelation:

genes_id1	genes_id of the query (string)
genes_id2	genes_id of the target (string)
sw_score	Smith-Waterman score between genes_id1 and genes_id2 (int)
bit_score	bit score between genes_id1 and genes_id2 (float)
identity	identity between genes_id1 and genes_id2 (float)
overlap	overlap length between genes_id1 and genes_id2 (int)
start_position1	start position of the alignment in genes_id1 (int)
end_position1	end position of the alignment in genes_id1 (int)
start_position2	start position of the alignment in genes_id2 (int)
end_position2	end position of the alignment in genes_id2 (int)
best_flag_1to2	best flag from genes_id1 to genes_id2 (boolean)
best_flag_2to1	best flag from genes_id2 to genes_id1 (boolean)
definition1	definition string of the genes_id1 (string)
definition2	definition string of the genes_id2 (string)
length1	amino acid length of the genes_id1 (int)
length2	amino acid length of the genes_id2 (int)

chiaramente per ognuno di questi campi è presente il corrispondente metodo get per recuperare le informazioni. (http://www.genome.jp/kegg/soap/doc/keggapi_javadoc/keggapi/SSDBRelation.html)

Passiamo ora a vedere una struttura diversa chiamata DBGET.

Si tratta di un metodo di recupero delle informazioni formattate come file di testo, detti anche flat-file. La definizione di flat-file di KEGG non è limitata ai soli file di testo ma si estende a immagini GIF (per i pathway di KEGG), grafica 3D per la struttura di proteine ecc.

La maggior parte degli attuali database biologici possono essere utilizzati in questa specifica.

Quando si passa un input ad un metodo di DBGET bisogna seguire un certo pattern:

dbname:identifier

ovvero nome del database (abbreviato) seguito dall'identificatore di ciò che ci interessa.

(per una lista completa delle abbreviazioni dei database si rimanda a: <http://www.genome.jp/dbget/>)

Il catalogo dei geni in KEGG considera anche come identificatore la combinazione organismo e gene:

organism:gene

Vediamo subito un esempio di codice e cosa ci restituisce:

```
import keggapi.*;

public class KeggEsDBGET {

    public static void main(String[] args) throws Exception {
        KEGGLocator locator = new KEGGLocator();
        KEGGPortType serv = locator.getKEGGPort();
        String result = serv.bget(args[0]);
        System.out.println(result);
    }
}
```

Provate a compilare ed eseguire questo codice con input “eco:b0004” dove in questo caso eco è l'organismo Escherichia coli e b0004 è l'id del gene.

Quello che vi ritorna il metodo bget è una stringa formattata che contiene tutte le informazioni relative all'argomento della ricerca.

```
ENTRY          b0004          CDS          E.coli
NAME           thrC, ECK0004, JW0003
DEFINITION     threonine synthase (EC:4.2.3.1)
ORTHOLOGY      K01733 threonine synthase [EC:4.2.3.1]
PATHWAY        eco00260 Glycine, serine and threonine metabolism
               eco00750 Vitamin B6 metabolism
               eco01100 Metabolic pathways
CLASS          Metabolism; Amino Acid Metabolism; Glycine, serine and threonine
               metabolism [PATH:eco00260]
               Metabolism; Metabolism of Cofactors and Vitamins; Vitamin B6
               metabolism [PATH:eco00750]
POSITION     3734..5020
MOTIF          Pfam: PALP
               PROSITE: DEHYDRATASE_SER_THR
DBLINKS        NCBI-GI: 16127998
               NCBI-GeneID: 945198
               RegulonDB: B0004
               EcoGene: EG11000
               UniProt: P00934
STRUCTURE      PDB: 1VB3
AASEQ          428
               MKLYNLKDHNEQVSFAQAVTQGLGKNQGLFFPHDLPEFSLTEIDEMLKLDVTRSAKILS
               AFIGDEIPQEILEERVRAAFAPAPVANVESDVGCLELFHGPTLAFKDFGGRFMAQMLTH
               IAGDKPVTILTATSGDTGAAVAHAFYGLPNVKVILYPRGKISPLQEKL FCTLGNNIETV
               AIDGDFDACQALVKQAFDDEELKVALGLNSANSINISRLLAQICYFEAVAQLPQETR NQ
               LVVSVPSGNFGDLTAGLLAKSLGLPVKRFIAATNVNDTVPRFLHDGQWSPKATQATLSNA
               MDVSQPNNWPRVEELFRRKIWQLKELGYAAVDDETTQTMRELKELGYTSEPAAVAYRA
               LRDQLNPGEYGLFLGTAHPAKFKESVEAILGETLDLPKELAEADLP LLSHNLPA DFAAL
               RKLMMNHQ
NTSEQ          1287
               atgaaactctacaatctgaaagatcacaacgagcaggtcagctttgcgcaagccgtaacc
               caggggttgggcaaaaatcaggggctgtttttccgcacgacctgccggaattcagcctg
               actgaaattgatgagatgctgaagctggattttgtcaccgcagtgcggaagatcctctcg
               gcgtttatttgtgatgaaatcccacaggaatcctggaagagcgcgtgcgcgcggcggtt
               gccttcccggctccggtcgccaatgttgaaagcagatgtcggttgtctggaattgttccac
               gggcaacgctggcatttaaagatttcggcggtcgctttatggcacaaatgctgacctat
               attgcgggtgataagccagtgaccattctgaccgcacctccggtgataccggagcgcca
               gtggctcatgctttctacggtttaccgaatgtgaaagtgggttatcctctatccacgaggc
               aaaatcagtcactgcaagaaaaactgttctgtacattgggcggaatatcgaaactgtt
               gccatcgacggcgatttctgatgcctgtcaggcgctggtgaagcaggcggttgatgatgaa
               gaactgaaagtggcgctaggggttaactcggctaactcgattaacatcagccgtttgtctg
               gcgcagatttgctactactttgaagctgttgccgcagctgccgcaggagacgcgcaaccag
               ctggttgtctcggtgccaagcggaacttcggcgatttgacggcggggtctgctggcgaag
               tcaactcggtctgccggtgaaacgttttattgtctgcgaccaacgtgaacgataccgtgcc
               cgtttctgcacgacggctcagtggtcacccaaagcgactcaggcgacgttatccaacgcg
               atggacgtgagtcagccgaacaactggccgcgtgtggaagagttgttccgccgcaaaatc
               tggcaactgaaagagctgggttatgcagccgtggatgatgaaaccacgcaacagacaatg
               cgtgagttaaaagaactgggctacacttcggagccgcacgctgccgtagcttatcgtgcg
               ctgctgatcagttgaatccaggcgaatatggcttgttccctcggcaccgcgcacatccggcg
               aaatttaaagagagcgtggaagcgaattctcggtgaaacgttgatctgcaaaaagagctg
               gcagaacgtgctgatttacccttgctttcacataatctgcccgccgattttgtctgcgttg
               cgtaaattgatgatgaatcatcagtaa
```

///

Chiaramente questa rappresentazione dei dati contiene un po' troppe informazioni e nessuna utilizzabile per qualche elaborazione successiva, è quindi necessario suddividere questa grossa stringa “ritagliando” solo le parti che ci interessano.

Esercizio:

Recuperare il valore di POSITION.

SUGGERIMENTO: utilizzate il metodo `indexOf()` per recuperare l'indice della parola POSITION e la posizione del primo punto dopo la parola POSITION con il metodo `indexOf(int ch, int fromIndex)` poi recuperate la sottostringa con il metodo `substring(start_index+12, end_index)`.

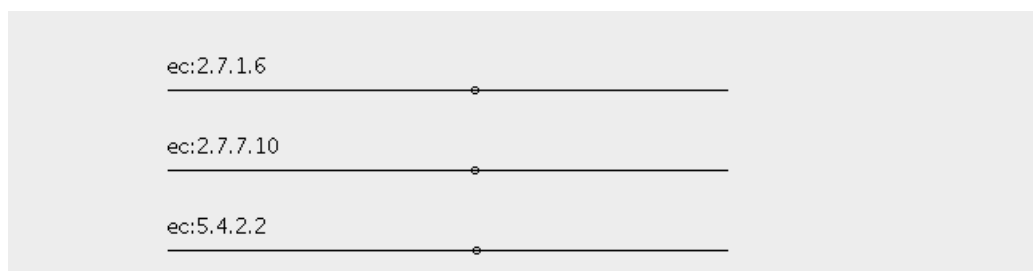
Esercizio finale:

Scrivere un programma che:

- 1) Recupera l'ID di un organismo letto come primo parametro sulla linea di comando;
ATTENZIONE: abbiamo già visto un possibile codice che lo fa, ma ad una voce inserita dall'utente possono essere associati più risultati, sarà quindi necessario gestire questa evenienza (anche semplicemente prendendo il primo risultato).
- 2) Dato l'id dell'organismo, realizzate un metodo che riceve come parametro un enzima e ritorna la posizione del gene corrispondente. L'enzima sarà passato al programma come secondo parametro sulla riga di comando. Provate con i seguenti enzimi :
 - ec:2.7.1.6
 - ec:2.7.7.10
 - ec:5.4.2.2

SUGGERIMENTO: usate il metodo `get_genes_by_enzyme()` per recuperare il gene e passate ciò che viene ritornato come argomento del metodo `bget()` già visto precedentemente.

- 3) Realizzate un secondo metodo che recupera la lunghezza del genoma:
SUGGERIMENTO: usate sempre `bget()` con la stringa “gn:”+id_organismo (vedere in Appendice B un esempio di output) e usate come delimitatore di `indexOf()` il carattere '\n' corrispondente a un a capo.
- 4) Tramite l'oggetto Canvas disegnate una retta che identifica il genoma e un pallino che identifica la posizione degli enzimi sul genoma (vedere Appendice A).
Attenzione alla scala e disegnate ogni enzima su una retta diversa, come in figura:



Provate usando come input “Lactobacillus johnsonii NCC 533” tra doppi apici. Se non usate i doppi apici o passate solo “johnsonii”, il vostro programma dovrebbe farvi scegliere tra altri organismi che contendono quella parola.

NOTA: il programma è solo un esempio prototipale e funziona bene solo con “Lactobacillus johnsonii NCC 533”; altri organismi potrebbero non contenere gli enzimi specificati oppure generare l'informazione della posizione con una sintassi diversa da quella vista e parserizzata negli esercizi.

Appendice A

Java e le primitive grafiche

Introduciamo brevemente le primitive che Java mette a disposizione del programmatore per la gestione della grafica elementare.

Queste primitive consentono da un lato di implementare GUI (Graphical User Interface) interattive con l'utente, dall'altro di giocare con i costrutti geometrici fondamentali e realizzare ogni sorta di disegno e rappresentazione grafica.

Vedremo dunque come implementare a livello applicativo la visualizzazione di un semplice grafico per la rappresentazione di figure geometriche elementari, al fine di visualizzare semplici rappresentazioni grafiche dei risultati.

Panoramica sul package AWT

Di seguito faremo riferimento all'Abstract Window Toolkit, la libreria contenente le classi e le interfacce fondamentali per la creazione di elementi grafici, inserita nelle API standard di Java.

I package fondamentali in cui si articola sono i seguenti:

- `java.awt` package principale che contiene le classi di tutti i componenti che possiamo utilizzare nella GUI, quali ad esempio `Frame`, `Panel`, `Button`, `Label`, `Checkbox`, `Canvas`, `Menu`, `MenuBar`, `TextArea` e molti altri
- `java.awt.event` fornisce le classi per la gestione degli “eventi”, ovvero il sistema che awt utilizza per passare il controllo al programmatore in seguito ad azioni avvenute sui componenti, come la pressione di un bottone, il passaggio del mouse su un componente, l'apertura di una finestra

La classe che rappresenta una “finestra” di interazione grafica con l'utente è la `java.awt.Frame`. Essa presenta fondamentalmente la classica barra del titolo e un bordo.

Ci concentreremo sull'uso di oggetti `Canvas`.

Canvas

`Canvas` è un componente che rappresenta un rettangolo vuoto dello schermo all'interno del quale è possibile disegnare oggetti geometrici elementari.

Vediamo subito un esempio per spiegare come utilizzare questo componente.

```
import java.awt.*;
import java.awt.event.*;

public class DrawExample extends Canvas {

    public void paint(Graphics g){
        g.drawRect(10, 50, 400, 1);
    }

    public static void main (String args[]){
        Canvas frame = new DrawExample();
        Frame finestra = new Frame();
```

```

        finestra.add(frame);
        finestra.setSize(450, 90);
        finestra.setResizable(false);
        finestra.setVisible(true);
    }
}

```

Come vedete la classe DrawExample estende Canvas, infatti per poter disegnare qualcosa sul canvas è necessario ridefinire il metodo paint.

L'oggetto passato come parametro al metodo paint è un oggetto Graphics, che definisce un contesto grafico con il quale disegnare sul Canvas.

Graphics mette a disposizione molti metodo per disegnare delle figure geometriche elementari quali rettangoli, cerchi, rette ecc.

Nell'esempio si utilizza il metodo drawRect(int x, int y, int width, int height);

i cui argomenti sono abbastanza autoesplicativi.

Analizziamo ora il main del progetto:

- Come prima cosa si crea un oggetto della classe che estende Canvas e quindi definisce cosa bisogna disegnare;
- L'oggetto Canvas appena creato deve essere messo in un Frame per poterlo visualizzare a video e quindi creiamo un oggetto di tipo Frame;
- Una volta creato il Frame basta impostare le dimensioni, togliere il resize della finestra e settare la visibilità.

Come già detto Graphics mette a disposizione un metodo per quasi tutte le forme geometriche elementari, si rimanda quindi alla Javadoc di Graphics per una lista completa:

(Graphics: <http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Graphics.html>)

Appendice B

Esempio di output della chiamata bget() con argomento “gn:”+id_organismo

```
ENTRY      T00158          Complete Genome
NAME       ljo, L.johnsonii, LACJ0, 257314
DEFINITION Lactobacillus johnsonii NCC 533
ANNOTATION manual
TAXONOMY   TAX:257314
    LINEAGE Bacteria; Firmicutes; Lactobacillales; Lactobacillaceae;
            Lactobacillus
DATA_SOURCE RefSeq
ORIGINAL_DB Nestle
CHROMOSOME Circular
    SEQUENCE RS:NC_005362
    LENGTH 1992676
STATISTICS Number of nucleotides:      1992676
            Number of protein genes:    1821
            Number of RNA genes:        97
REFERENCE  PMID:14966310
    AUTHORS Pridmore RD, et al.
    TITLE   The genome sequence of the probiotic intestinal bacterium
            Lactobacillus johnsonii NCC 533.
    JOURNAL Proc Natl Acad Sci U S A : (2004)
///
```