Biojava 使用指南及部分程序开发

1. 简介.....3
2. 下载与安装.....3
3. 功能简介.....3
3.1 成分表(alphabets)和标记(symbol).....4
3.1.1 如何得到 DNA,RNA 或蛋白质的成分表？.....4
3.1.2 如何用自定义的标记建立自定义的成分表?.....5
3.1.3 如何建立杂交产物成分表(cross product alphabet),例如密码字成分表(codon alphabet)?.....6
3.1.4 如何从杂交产物成分表(cross product alphabet)中分解出他们的组成标记(component symbol)?.....6
3.1.5 如何判别两个成分表或两个标记是否相同?.....7
3.1.6 如何建立一个多义标记(ambiguous symbol),例如 Y 或 R?.....8
3.2 基本序列操作.....8
3.2.1 如何从字串中创建一条序列对象以及将其写回一条字串?.....8
3.2.2 如何从一条序列中得到子序列?.....9
3.2.3 如何将一条 DNA 序列转录成一条 RNA 序列？.....10
3.2.4 如何改变序列的名字？.....11
3.2.5 如何编辑一条序列?.....11
3.3 翻译(translation).....13
3.3.1 如何将一条 DNA 或 RNA 或标记链翻译成蛋白质?.....13
3.3.2 如何将单个密码子翻译成单个氨基酸？.....13
3.3.3 如何使用非标准的翻译表？.....14
3.4 序列输入输出(sequence I/O).....15
3.4.1 如何将一条序列以 Fasta 格式输出？.....15
3.4.2 如何从一个 Fasta 文件中读取序列？.....16
3.4.3 如何读取一个 GenBank,SwissProtm,EMBL 文件？.....17
3.4.4 如何将 ABI 序列转化为 BioJava 序列?(待测试).....19
3.5 注释(annotation).....20
3.5.1 如何列出序列中的注释？.....20
3.5.2 如何根据物种属性过滤序列？.....22

3.6 位置和特征(location and feature) 23
3.6.1 如何定制一个点位置(PointLocaiton)？ 23
3.6.2 如何定制一个范围位置(RangeLocation)？ 23
3.6.3 如何处理环状位置？ 24
3.6.4 如何创建一个特征？ 25
3.6.5 如何根据类型来筛选特征？ 25
3.6.6 如何从序列中删除特征？ 26
3.7 BLAST 和 FASTA（待测试） 27
3.7.1 如何设置一个 BLAST 解析器解析 BLAST 结果？ 27
3.7.2 如何设置一个 FASTA 解析器解析 FASTA 搜索结果？ 28
3.7.3 如何从搜索结果中提取信息？ 29
3.8 计数和分布(count and distribution) 29
3.8.1 如何计算序列中的残基数？ 29
3.8.2 如何将计数转化为一个分布？ 31
3.8.3 如何从一个分布得到一条随机序列？ 32
3.8.4 如何计算一个分布中的信息量或熵？ 32
3.8.5 如何能找到一种简单的方法来判断两种分布是否具有相同的权重? 33
3.8.6 如何对一个自定义的成分表创建一个 N 阶分布(order N distribution)? 34
3.8.7 如何将一个分布以 XML 格式输出? 35
3.9 权重矩阵和动态规划(weight matrix and dynamic programming) 36
3.9.1 如何使用一个权重矩阵来寻找一个模体？ 36
3.9.2 如何创建一个隐马模型序列谱(Profile HMM)?(待测试) 37
3.9.3 如何建立一个自定义的隐马模型(HMM)? 39
3.10 用户界面(user interfaces) 39
3.10.1 如何以树状结构展示特征和注释？ 39
3.10.2 如何图形化一条序列? 40
3.10.3 如何展示序列标尺？ 40
3.10.4 如何图形化一个特征？ 41
3.11 如何设置 BioSQL?(待测试) 41
4. 总结 45

1. 简介

和生物信息学沾边的编程语言浩如烟海，BioPHP、BioJava、BioPerl、BioPython、BioRuby 等等……据 Bioinformatics.org 上最新的投票显示（http://www.bioinformatics.org/poll/index.php?dispid=16）：BioJava 以 17%的选择率位列第三，仅次于 Python 和 PERL。如果考虑到在其他领域的受欢迎程度和程序执行速度，则 Java 又大幅胜出。那么 BioJava 能做些什么呢？目前，其功能主要覆盖以下几个方面：基因组（序列转换、基因注释、BLAST&FASTA）；蛋白质组（等电点计算、蛋白结构预测、序列比对）、一些常用算法（遗传算法、HMM、动态规划等）和 BioSQL（生物学数据库支持，包括一般的数据库类型、序列数据库和 Ontology 数据库）。就我目前使用过的极为有限的编程工具而言，与 R 相比，优势在于嵌入到 Java 程序更为方便，缺点是不容易上手（需要先熟悉 Java）；与 Matlab 相比优势在于开源，与 Perl 相比执行效率更高。

2. 下载与安装

这个是已经开发的开源 jar 包，不需要安装，只需要在需要他的时候导入就可以了。下载可以去这个地址下载：http://www.biojava.org/download/ 我下载的是最新版本的 3.0.3。这个用来测试后边那个 blast，其他的程序都是在 bi15 版本下测试的。所以大家下载的时候注意一下，他是有两个模式的版本，不太一样。

3. 功能简介

下边我会罗列他的一些功能，用简单的代码表示，当然这只是他的其中一些而已，更多得功能等待着去摸索。而且我只给出核心代码，完整代码程序最后会单独给出，这里就不罗列了。

声明：以下的代码原创不是我，我只是参考一个网站上的，但是他的上边有很多都是不完全的或者是有很多错误的，下边的只要不是标注待测试的，我都是改善过之后的代码，都是能够运行出结果的。我会给出一个代码的例子，大家可以对应着每个例子去参考一下，代码上我也标注过了。再次声明，原创不是我，因此希望大家都是以学术参考为前提。例子的原网址是：

http://www.cbi.pku.edu.cn/chinese/documents/PUMA/biojava/index-cn.html 建议大家可以看一看，不要轻易直接使用他的代码，因为很多都是错误和不完善的，很多他只是给出一个思想，但是我给出的代码基本上是没有错误和基本完善的，但是也不完全保证。

另外，凡是没有完整代码的我都是参考 Biojava 的 API 文档，文档地址为：

http://www.biojava.org/docs/api1.8.2/ 和

http://www.biojava.org/docs/api/index.html

主要参考第一个。

3.1 成分表(alphabets)和标记(symbol)

3.1.1 如何得到 DNA,RNA 或蛋白质的成分表？

在 BioJava 中，成分表（alphabets)是标记（symbol)的集合。（例如，DNA 就是一种成分表，其中 a,c,g,t 是标记，DNA 是这四种标记的集合。相似地，RNA 由 a,c,g,u 四种标记组成，蛋白质由二十种氨基酸标记组成）。普通的生物成分表（DNA，RNA，蛋白质等等）由 BioJava 的 AlphabetManager 负责初始化和注册，可以通过名字进行访问。DNA，RNA，蛋白质成分表也可以分别通过 DNATools,RNATools 和 ProteinTools 的静态方法方便地访问。

参考代码如下：

```
Alphabet dna, rna, prot;
// 使用名字来创建 DNA 成分表
dna = AlphabetManager.alphabetForName("DNA");
//输出 DNA 的各个标记，即 A,G,C,T，RNA 和蛋白质类似
FiniteAlphabet fa=(FiniteAlphabet)(dna);
for (Iterator i = fa.iterator();i.hasNext();)
{
    Symbol sym = (Symbol)i.next();
    System.out.println(sym.getName());
}
// 使用名字来创建 RNA 成分表
rna = AlphabetManager.alphabetForName("RNA");
// 使用名字来创建蛋白质成分表
prot = AlphabetManager.alphabetForName("PROTEIN");
// 创建包含*终止标记的蛋白质成分表
prot = AlphabetManager.alphabetForName("PROTEIN-TERM");
// 从 Tools 类中创建这些成分表
```

```
dna = DNATools.getDNA();
rna = RNATools.getRNA();
prot = ProteinTools.getAlphabet();
// 或创建含有终止标记的蛋白质成分表( 注意 getAlphabet()和 getTAlphabet() )
prot = ProteinTools.getTAlphabet();
```

3.1.2 如何用自定义的标记建立自定义的成分表?

本例子讲述如何创建一个“二进制”的成分表，它包括两种标记：0 或 1。用户可以制定自己的标记和成分表，然后可以用来创建标记链，序列，分布等等。

参考代码：

```
// 创建一个没有注释的标记“0”
        Symbol zero = AlphabetManager.createSymbol("zero", Annotation.EMPTY_ANNOTATION);

        // 创建一个没有注释的标记“1”
        Symbol one = AlphabetManager.createSymbol("one", Annotation.EMPTY_ANNOTATION);
        // 创建一个集合来存储标记
        Set symbols = new HashSet();
        symbols.add(zero);
        symbols.add(one);
        // 创建“二进制成分表
        FiniteAlphabet binary = new SimpleAlphabet(symbols,"Binary");
        // 遍历所有标记，验证没有问题
        for (Iterator i = binary.iterator();i.hasNext();)
        {
            Symbol sym = (Symbol)i.next();
            System.out.println(sym.getName());
        }
        // 通常使用成分表管理器（AlphabetManager)注册新创建的成分表
        AlphabetManager.registerAlphabet(binary.getName(),binary);
        // 新创建的成分表使用“Binary”作为名字，这样你可以象先前那样规范的通过名字来创建一个成分表实例
        Alphabet alpha = AlphabetManager.alphabetForName("Binary");
        // 检查规范性
        System.out.println(alpha == binary);
```

3.1.3 如何建立杂交产物成分表(cross product alphabet),例如密码字成分表(codon alphabet)?

杂交产物成分表由多个其他成分表生成。目的是将两个或者多个标记包装成单个的“杂交产物”标记。例如，使用三个 DNA 成分表杂交可以将密码子（三联密码子）包装成一个标记。这样，你可以使用这个密码子标记进行统计计数或者在一种分布中使用这个密码子标记。

杂交产物成分表可以通过名字创建（如果组成成分表在成分表管理器中注册了的话），也可以创建一个组成成分表的列表，然后从这个列表中创建杂交产物成分表。这两种方法在下面的例子中都有使用。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 从一个列表中创建杂交产物成分表

        List list = Collections.nCopies(3, DNATools.getDNA());
        Alphabet codon = AlphabetManager.getCrossProductAlphabet(list);
        // 用名字得到同样的杂交产物成分表
        Alphabet codon2=AlphabetManager.generateCrossProductAlphaFromName("(DNA x DNA x DNA)");
        // 验证这两个杂交产物成分表一致
        System.out.println(codon==codon2);
---------------------------------------------------------------------------------------------------------------
```

3.1.4 如何从杂交产物成分表(cross product alphabet)中分解出他们的组成标记(component symbol)?

杂交产物成分表用来将一组标记包装成单个标记。这有利于处理象密码子这样的标记。然而，有时候我们希望将这些标记转变回它们的组成标记，下面的例子说明了如何完成这个任务。

杂交产物成分表中的标记实现了原子标记（atomic symbol)接口。前缀“原子”显示这个标记不可再分。所以有人就会问道，“如何将不可再分的标记分成它的组成标记？”。原子标记的完备定义是它不能再被分解成简单的标记，而这个简单标记却又和原子标记处在相同成分表中。这样，杂交产物成分中的标记并不和它的组成标记处于同一个成分表中。（例如,atg,acc att 是密码子标记，由 DNAxDNAxDNA 杂交生成，但在密码子成分表中并不包含 a,g,t,c。所以其原子性仍成立）密码子对应（DNAxDNAxDNA）成分表，而它的组成标记对应 DNA 成分表。

和基本标记（Basis symbol)相比较，基本标记分解的组成标记可以和基本标记处于同一个成分表中。

用这种方法，一个基本标记可以是多义的。更多讨论参见基本标记。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 创建密码子成分表

        List list = Collections.nCopies(4, DNATools.getDNA());
        Alphabet alph = AlphabetManager.getCrossProductAlphabet(list);
        // 从成分表中取得第一个原子标记
        Iterator iter = ((FiniteAlphabet)alph).iterator();
        AtomicSymbol codon = (AtomicSymbol)iter.next();
        // 将它(第一个标记）的组成标记列举出来
        List symbols = codon.getSymbols();
---------------------------------------------------------------------------------------------------------------
```

3.1.5 如何判别两个成分表或两个标记是否相同?

在 Biojava 中，无论字母表和标记是通过何种方式建立的，相同的字母表和标记都是规范一致的。这意味着如果在不同时间创建的两个 DNA 字母表（或者是来自那些字母表中的标记）对象是相等的，通过调用.equals()方法和==操作符都可以得到这个结果。这对来自 PROTEIN 字母表和 PROTEIN-TERM 字母表或 IntergerAlphabet 和 SubIntegerAlphabet 的标记都成立。上述甚至对于运行在不同虚拟机上的对象都成立（这要感谢 JAVA 的序列化技术），这意味着 Biojava 可以进行 RMI 远程调用（这点对于开发分布式系统非常有好处）。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 通过两种不同的方法得到 DNA 字母表
        Alphabet a1 = DNATools.getDNA();
        Alphabet a2 = AlphabetManager.alphabetForName("DNA");

        // 判断是否相同
        System.out.println("equal: "+ a1.equals(a2));
        // 判断是否一致
        System.out.println("canonical: "+ (a1 == a2));
---------------------------------------------------------------------------------------------------------------
```

3.1.6 如何建立一个多义标记(ambiguous symbol),例如 Y 或 R?

IBU 定义了标准的多义标记。例如 Y 代表 C 或 T, R 代表 G 或 C, N 代表任何一种碱基。Biojava 用基本标记(BasisSymbol)来表示这些标记。基本标记对象可以包含一个或多个组份标记，而这些组份标记可以和基本标记在同一字母表中。通过这样的方法，我们可以实现标记的多义性。

通常一个多义标记可以在包含其组份标记的字母表中调用 getAmbiguity(Set symbols)方法获得。例如将 symbols 参数设为 DNA 字母表中的标记 C 和 T 组成的集合(Set),就可以获得多义标记 Y。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 获得 DNA 字母表
        Alphabet dna = DNATools.getDNA();

        // 创建 Y 标记
        Set symbolsThatMakeY = new HashSet();
        symbolsThatMakeY.add(DNATools.c());
        symbolsThatMakeY.add(DNATools.t());
        Symbol y = dna.getAmbiguity(symbolsThatMakeY);

        // 打印基本标记 Y 的信息
        System.out.println("Formal name of 'Y' is: "+y.getName());
        System.out.println("Class type of 'Y' is: "+y.getClass().getName());
---------------------------------------------------------------------------------------------------------------
```

3.2 基本序列操作

3.2.1 如何从字串中创建一条序列对象以及将其写回一条字串?

很多时候我们将序列用一个由字符组成的字串表示，如 "atgccgtggcatcgaggcatatagc". 这是表示复杂生物序列的一种非常方便简洁的方法。Biojava 使用 SymbolLists 和 Sequences 将这些生物序列表示成对象。Sequences 继承 SymbolLists，提供特殊的方法来储存诸如序列名称，

序列特征等等。你可以认为 Sequence 是一种 SymbolList.，Sequence 和 SymbolList 中序列不是以字串形式存储的。Biojava 使用不同的 Symbol 对象（来自不同的字母表）来区别不同的生物序列残基。用这种方法可以容易的识别序列是 DNA 还是 RNA，或者标记 A 是 DNA 中的标记 A 还是 RNA 中的标记 A。标记，标记链表（SymbolList)和字母表详情在此。问题的关键是要有一种方法能够使程序员很方便的在字串表示法和对象表示法间转换。Biojava 使用 Tokenizers（标识符）能够从文本中读取并解析成 Biojava 中的 Sequence 或 SymbolList 对象，反之亦然。例如，可以从 DNATools，RNATools, ProteinTools 中调用一种静态方法进行上述操作。

参考代码：

```
---------------------------------------------------------------------------------------------------------------------

/*tring to SymbolList*/
// 从字串中创建 DNA SymbolList
                SymbolList dna = DNATools.createDNA("atcggtcggctta");
                // 从字串中创建 RNA SymbolList
                SymbolList rna = RNATools.createRNA("auugccuacauaggc");
                // 从字串中创建蛋白质 SymbolList
                SymbolList prot = ProteinTools.createProtein("AGFAVENDSA");
---------------------------------------------------------------------------------------------------------------------
/*String to Sequence*/
// 创建一个叫 dna_1 的 DNA 序列
                Sequence dna = DNATools.createDNASequence("atgctg", "dna_1");
                // 创建一个叫 rna_1 的 RNA 序列
                Sequence rna = RNATools.createRNASequence("augcug", "rna_1");
                // 创建一个叫 prot_1 的 Protein 序列
                Sequence prot = ProteinTools.createProteinSequence("AFHS", "prot_1");
---------------------------------------------------------------------------------------------------------------------
/*SymbolList to String*/
                SymbolList sl = null;
                /*
                在这里实例化 sl
                */
                // 将 sl 转化成字串
                String s = sl.seqString();
---------------------------------------------------------------------------------------------------------------------
```

3.2.2 如何从一条序列中得到子序列?

给定一条序列，我们也许只关心前 10 个碱基或者我们想得到序列中的一段。你也许想打印

一条子序列至输出流，例如 STDOUT。如何做到这些？

Biojava 使用生物学坐标系统识别碱基。第一个碱基索引为 1，最后一个碱基索引为序列长度。注意这里和计算机中字串的索引不同（以零开始）。如果你的读取超过了 1 到序列长度的范围，会抛出异常。

参考代码：

```
SymbolList symL = null;

        //generate an RNA SymbolList
        try
        {
              symL = RNATools.createRNA("auggcaccguccagauu");
        }
        catch (IllegalSymbolException ex)
        {
              ex.printStackTrace();
        }

        //get the first Symbol
        Symbol sym = symL.symbolAt(1);

        //get the first three bases
        SymbolList symL2 = symL.subList(1,3);

        //get the last three bases
        SymbolList symL3 = symL.subList(symL.length() - 3, symL.length());

        //print the last three bases
        String s = symL.subStr(symL.length() - 3, symL.length());
        System.out.println(s);
```

3.2.3 如何将一条 DNA 序列转录成一条 RNA 序列？

在 Biojava 中，DNA 和 RNA 序列以及标志链(SymbolList)使用不同的字母表。你可以使用 RNATools 的静态方法 transcribe()将 DNA 转录成 RNA。

参考代码：

```
// 创建一条 DNA 标志链
```

```
SymbolList symL = DNATools.createDNA("atgccgaatcgtaa");

// 转录成 RNA
symL = DNATools.transcribeToRNA(symL);
```

3.2.4 如何改变序列的名字？

大多数 Biojava 序列对象是不可改变的。这样的安全特性能避免对数据完整性的破坏。这样，在序列对象中就没有 setName()这样的方法。有一种办法是用原始序列作为参数创建原始序列对象的视图。

通过使用序列视图这样的包装器能够实现很多不能直接对原始序列对象调用的方法，例如改变序列的名字。

参考代码：

```
Sequence seq = DNATools.createDNASequence("atgcgctaggctag","gi|12356|ABC123");

// 创建一个序列视图 改变它的名字
ViewSequence seq2 = new ViewSequence(seq,"ABC123");
// 输出 FASTA 格式证明名字改变了
SeqIOTools.writeFasta(System.out,seq2);
```

3.2.5 如何编辑一条序列?

有时候你可能希望改变序列或标志链的标志次序.例如你可能希望能删除或者插入或者重写 DNA 序列中的碱基.

Biojava 标志链有一个 edit(Edit e)方法,能够以 Edit 对象为参数编辑标志链.Edit 对象需要编辑起始位置,编辑数量为参数来编辑序列.

需要注意的是:很多 Biojava 序列或标志链不允许编辑操作,因为这样会破坏序列的注释和特征.最好的策略是拷贝序列一个副本,然后对这个副本进行编辑操作.下面的程序体现了这点.

参考代码：

```
// 你不能真正的编辑这条序列
        Sequence seq = DNATools.createDNASequence("atggct", "seq");
```

```
// 所以需要这条序列的一个拷贝,利用拷贝构造器
SimpleSymbolList syms = new SimpleSymbolList(seq);

// 将"cc"加到序列末尾,覆盖 0 个碱基
Edit e = new Edit(seq.length()+1, 0, DNATools.createDNA("cc"));

// 应用 Edit 对象
syms.edit(e);

// 现在序列是"ttatggctcc"
System.out.println(syms.seqString());

// 将"aca"插入第四个位置,覆盖 0 个碱基
e = new Edit(4,0,DNATools.createDNA("aca"));
syms.edit(e);

// 用"ggg"覆盖在第二个位置开始的三个碱基
e = new Edit(2,3,DNATools.createDNA("ggg"));
syms.edit(e);

// 现在应该是"tgggcatggctcc"
System.out.println(syms.seqString());

// 删除开始的五个碱基
e = new Edit(1,5, SymbolList.EMPTY_LIST);
syms.edit(e);

// 现在应该是"atggctcc"
System.out.println(syms.seqString());

// 一个更复杂的例子
// 用"aa"覆盖位置 2,3 的碱基然后插入"tt"
e = new Edit(2,2,DNATools.createDNA("aatt"));
syms.edit(e);

// 现在应该是"aaattgctcc"
System.out.println(syms.seqString());
```

3.3 翻译(translation)

3.3.1 如何将一条 DNA 或 RNA 或标记链翻译成蛋白质?

要将 DNA 翻译成蛋白必须执行下列步骤：

1 转录成 RNA

2 获得标志链的密码子（三联子）

3 翻译成蛋白

大部分可以通过使用 Biojava 的 Tools 类的静态方法实现。下面的程序展示了这个过程。当然了，如果你的序列已经是 RNA，那么就不用转录步骤。

注意：如果你对一条长度非三的倍数的序列或标志链创建三联子视图（triplet view)，IllegalArgumentException 将会被抛出。参看“如何得到子序列“将会提示你如何得到序列的一部分，用这一部分进行翻译。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
//创建一个 DNA 序列
        SymbolList symL = DNATools.createDNA("atggccattgaatga");

        //转录成 RNA
        symL = DNATools.transcribeToRNA(symL);

        //把 RNA 翻译成蛋白质
        symL = RNATools.translate(symL);
---------------------------------------------------------------------------------------------------------------
```

3.3.2 如何将单个密码子翻译成单个氨基酸？

通常的翻译的例子展示了如何使用 RNATools 将 RNA 标志链翻译成蛋白质，但是具体细节隐藏在 translate()方法中。如果你想将单个的密码子翻译成氨基酸，那么你就要明白具体的细节了。

实际上有很多方法实现上述目的，这里我仅仅展示其中的一种。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 创建一个以密码子为标志的复合字母表
        Alphabet a = AlphabetManager.alphabetForName("(RNA x RNA x RNA)");
```

```
// 获得一个翻译表
TranslationTable table = RNATools.getGeneticCode(TranslationTable.UNIVERSAL);
try
{
    // 创建一个密码子
    SymbolList codon = RNATools.createRNA("UUG");
    // 得到一个密码子标志
    Symbol sym = a.getSymbol(codon.toList());
    // 翻译成氨基酸
    Symbol aminoAcid = table.translate(sym);
    // 证明得到的标志是氨基酸
    ProteinTools.getAlphabet().validate(aminoAcid);
}
catch(IllegalSymbolException ex)
{
    ex.printStackTrace();
}
```

3.3.3 如何使用非标准的翻译表？

在一般的翻译例子中我们使用 RNATools 中的方法 translate()就能够很容易的进行翻译。这对于使用一般的通用翻译表来说非常合适。但有的时候，你可能使用一些诸如叶绿体翻译表等偏僻的翻译表。幸运的是，这在 Biojava 中不是难题。RNATools 有一个静态的方法 getGeneticCode(String name) 能够使你通过名字来获取翻译表。

以下就是 Biojava 提供的翻译表：

FLATWORM_MITOCHONDRIAL

YEAST_MITOCHONDRIAL

ASCIDIAN_MITOCHONDRIAL

EUPLOTID_NUCLEAR

UNIVERSAL

INVERTEBRATE_MITOCHONDRIAL

BLEPHARISMA_MACRONUCLEAR

ALTERNATIVE_YEAST_NUCLEAR

BACTERIAL

VERTEBRATE_MITOCHONDRIAL

CLIATE_NUCLEAR

MOLD_MITOCHONDRIAL

ECHINODERM_MITOCHONDRIAL

这些都是有效的参数，可以被 RNATools.getGeneticCode(String name)使用，这些翻译表也可以通过 TranslationTools 类

中的静态字串获得。

下面的例子使用 Euplotid Nuclear 翻译表（UGA->Cys)。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 获得 Euplotid Nuclear 翻译表
        TranslationTable eup = RNATools.getGeneticCode(TranslationTable.EUPL_NUC);
        try
        {
            // 创建一条包含"tga"密码子的 DNA 序列
            SymbolList seq = DNATools.createDNA("atgggcccatgaaaaggcttggagtaa");
            // 转录成 RNA
            seq = DNATools.transcribeToRNA(seq);
            // 在 RNATools.translate()中，将 RNA 序列视为密码子序列
            seq = SymbolListViews.windowedSymbolList(seq,3);
            // 使用 Euplotid Nuclear 翻译表进行翻译
            SymbolList protein = SymbolListViews.translate(seq,eup);
            // 输出蛋白质
            System.out.println(protein.seqString());
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
---------------------------------------------------------------------------------------------------------------
```

3.4 序列输入输出(sequence I/O)

3.4.1 如何将一条序列以 Fasta 格式输出？

Fasta 格式是一种相当标准的符合生物信息学的输出，很容易读取。Biojava 中有一个 SeqIOTools 的类提供很多方便的静态方法，能够完成很多通用的符合生物信息学的输入输出任务。下面的例子展示如何将一条序列甚至整个 SequenceDB 以 Fasta 格式输出到一个输出流如 System.out 中。SeqIOTools 中所有 WriteXX()方法都一个输出流作为参数。用这样的方

法，你将可以利用管道方式把新格式化的序列写到文件中或传递到另外一个方法中或者是标准输出，错误输出等等。

SeqIOTools 在 org.biojava.bio.seq.io 包中。

参考代码：

```
// 创建一个 SequenceDB 接口实例
        SequenceDB db = new HashSequenceDB();
        // 将序列添加到序列库中
        Sequence seq1=SequenceTools.createDummy("AGTCAGTTTCTTC", "abc123");
        Sequence seq2=SequenceTools.createDummy("GCCAGGATGGGGC", "abc456");
        db.addSequence(seq1);
        db.addSequence(seq2);
        // 以 Fasta 格式打印序列库
        SeqIOTools.writeFasta(System.out, db);

        //从 SequenceIterator 中打印
        //SeqIOTools 中的很多 ReadXXX()方法读取了一个文件中的所有序列后返回一个 SequenceIterator 能够遍历所有序列。SeqIOTools 中大多数 WriteXXX()方法都有一个以 SequenceIterator 为参数的版本。
        BufferedReader br=new BufferedReader(new FileReader("F:/test/query.fasta"));
        SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava("fasta","DNA", br);
        // 以 Fasta 格式输出 （也可以写到出输出流中）
        SeqIOTools.writeFasta(System.out,iter);

        //打印单个序列
        SeqIOTools.writeFasta(System.out,seq1);
```

3.4.2 如何从一个 Fasta 文件中读取序列？

一个重要并且常用的 I/O 任务是将序列从文本文件读到内存中。SeqIOTools 提供很多静态方法能够将文件读到 Biojava 中。

其实这个，在上个程序中就已经给出说明了，这里再解释一次。

参考代码：

```
/*第一种方法*/
// 创建文件输入
```

```
String filename = file;
BufferedInputStream is = new BufferedInputStream(new FileInputStream(filename));
// 获得正确的字母表
Alphabet alpha = AlphabetManager.alphabetForName(alp);
// 从文件中获得代表所有序列的序列库
SequenceDB db = SeqIOTools.readFasta(is,alpha);
```

```
/*第二种方法*/
//准备一个缓冲
BufferedReader br = new BufferedReader(new FileReader(file));
String format = type;
String alphabet = alp;
// 获得一个序列遍历器
// SeqIOTools.fileToBiojava()返回一个对象。如果文件是比对的格式如 MSF,则返回一个比对对象。
SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(format,alphabet,br);
```

3.4.3 如何读取一个 GenBank,SwissProtm,EMBL 文件？

SeqIOTools 类包含了读取 GenBank,SwissProt,EMBL 文件的方法。因为文件中包含了不止一条序列，所以 SeqIOTools 返回一个序列遍历器（SequenceIterator)能够遍历所有的序列。这种模型最吸引人的特点是能够在需要的时候再创建序列对象，其他时候仅仅解析序列。这样能够用很少的资源处理大量的序列集合。

文件中序列的信息被存为序列的注释对象，位置信息被存为序列的特征对象。

参考代码：

```
/*读 GeneBank 文件*/
BufferedReader br = null;
    try
    {
        br = new BufferedReader(new FileReader(file));
    }
    catch(FileNotFoundException ex)
    {
        ex.printStackTrace();
        System.exit(-1);
    }
```

```
        // 读取 GenBank 文件
        SequenceIterator sequences = SeqIOTools.readGenbank(br);
        // 遍历所有序列
        while(sequences.hasNext())
        {
            try
            {
                Sequence seq = sequences.nextSequence();
            }
            catch(BioException ex)
            {
                ex.printStackTrace();
            }
            catch(NoSuchElementException ex)
            {
                ex.printStackTrace();
            }
        }
```

```
/*读取 SwissPort 文件*/
//***** same as example above *****
        SequenceIterator sequences = SeqIOTools.readSwissprot(br);
        //***** same as example above *****
```

```
/*读取 Empl 文件*/
//***** same as example above *****
        SequenceIterator sequences = SeqIOTools.readEmbl(br);
        //***** same as example above *****
```

```
/*通用阅读器 1*/
// 这个程序第一个参数是文件名，第二个参数代表文件类型的常量：
    // UNKNOWN = 0;
    // FASTADNA = 1;
    // FASTAPROTEIN = 2;
    // EMBL = 3;
    // GENBANK = 4;
    // SWISSPROT = 5;
    // GENPEPT = 6;
    // MSFDNA = 7;
    // FASTAALIGNDNA = 9;
    // MSFPROTEIN =10;
    // FASTAALIGNPROTEIN= 11;
    // MSF = 12;
BufferedReader br = new BufferedReader(new FileReader(file));
```

```
            int fileType = Integer.parseInt(type);
            SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(fileType,br);
```

```
/*通用阅读器 2*/
// 这个程序读取任何 SeqIOTools 支持的文件类型，共有三个参数。
    // 第一个是文件名，第二个是文件类型,第三个是字母表.
    // 支持的文件类型:(大小写敏感)
    // FASTA, EMBL, GENBANK, SWISSPROT (swiss),GENPEPT
    // 支持的序列类型:(大小写敏感)
    // DNA, AA(Protein)
// 准备缓存
            BufferedReader br = new BufferedReader(new FileReader(file));
            // 文件类型
            String format = type;
            // 字母表
            String alpha = alp;

            // 从文件中获得能够遍历所有序列的遍历器.SeqIOTools.fileToBiojava()方法返回一个对象.
            // 如果文件是比对格式例如 MSF,则返回比对对象.否则,返回一个遍历器对象.
            SequenceIterator iter = (SequenceIterator) SeqIOTools.fileToBiojava(format,alpha,br);
            // 处理序列
            SeqIOTools.writeFasta(System.out, iter);
```

3.4.4 如何将 ABI 序列转化为 BioJava 序列?(待测试)

很多生物信息学工作从读取测序仪的 DNA 片断开始.常用的是 ABI 测序仪的输出.Biojava 包括了一个 ABITrace 类能够解析 ABITrace 文件或 URL 或者是存储他的 byte[]数组.下面的程序从 ABITrace 文件中创建 Biojava 序列.

Biojava1.3 提供了更通用的色谱接口,支持 ABI 和 SCF 文件.下面的程序使用了这个接口.

参考代码:

这个代码是没有经过测试的,因为暂时没有找到 ABITrace 文件。

```
读取 ABI
File traceFile = new File("文件地址");

        //序列名
        String name = traceFile.getName();
        //读取 trace
        ABITrace trace = new ABITrace(traceFile);
```

```
        //抽取标志
        SymbolList symbols = trace.getSequence();
        //创建完整的序列
        Sequence seq = new SimpleSequence(symbols, name, name,
Annotation.EMPTY_ANNOTATION);
        //输出到标准输出
        SeqIOTools.writeFasta(System.out, seq);
```

通用模式

```
File traceFile = new File("");
        //序列名
        String name = traceFile.getName();
        //读取 trace
        Chromatogram trace = ChromatogramFactory.create(traceFile);
        //抽取标志
        SymbolList symbols = ChromatogramTools.getDNASequence(trace);
        //创建完整序列
        Sequence seq = new SimpleSequence(symbols, name, name,
Annotation.EMPTY_ANNOTATION);
        //输出到标准输出
        SeqIOTools.writeFasta(System.out, seq);
```

3.5 注释(annotation)

3.5.1 如何列出序列中的注释？

当你读取象 GenBank 或 EMBL 这样的序列注释文件时，文件提供的不仅仅是序列本身还有一些更细节的序列信息。如果这个信息拥有位置的话，就可以当作是特征。如果这个信息是很通用的信息比如说是物种名称的话，就可以当作是注释。Biojava 注释（Annotation)对象有些象图（Map)对象，它包括键值映射。

参考代码：

```
// 读取 EMBL 纪录
        BufferedReader br = new BufferedReader(new FileReader("F:/test/test"));
        SequenceIterator seqs = SeqIOTools.readEmbl(br);
        // 对于每条序列列出它的注释
        while(seqs.hasNext())
        {
            Annotation anno = seqs.nextSequence().getAnnotation();
```

```
            // 打印每个键值对
            for(Iterator i = anno.keys().iterator();i.hasNext(); )
            {
                Object key = i.next();
                System.out.println(key+" : "+ anno.getProperty(key));
            }
        }
```

针对上边的程序，我给出一个 EMBL 文件的开头部分，这个程序能够读出他的属性。

ID AY130859 standard; DNA; HUM; 44226 BP.
XX
AC AY130859;
XX
SV AY130859.1
XX
DT 25-JUL-2002 (Rel. 72, Created)
DT 25-JUL-2002 (Rel. 72, Last updated, Version 1)
XX
DE Homo sapiens cyclin-dependent kinase 7 (CDK7) gene, complete cds.
XX
KW .
XX
OS Homo sapiens (human)
OC Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia;
OC Eutheria; Primates; Catarrhini; Hominidae; Homo.
XX
RN [1]
RP 1-44226
RA Rieder M.J., Livingston R.J., Braun A.C., Montoya M.A., Chung M.-W.,
RA Miyamoto K.E., Nguyen C.P., Nguyen D.A., Poel C.L., Robertson P.D.,
RA Schackwitz W.S., Sherwood J.K., Witrak L.A., Nickerson D.A.;
RT ;
RL Submitted (11-JUL-2002) to the EMBL/GenBank/DDBJ databases.
RL Genome Sciences, University of Washington, 1705 NE Pacific, Seattle, WA
RL 98195, USA
XX
CC To cite this work please use: NIEHS-SNPs, Environmental Genome
CC Project, NIEHS ES15478, Department of Genome Sciences, Seattle, WA
CC (URL: http://egp.gs.washington.edu).
//

注意：EMBL 文件的序列是以“//”结尾的，所以最后要在补上一个结尾符，要不会出错。

3.5.2 如何根据物种属性过滤序列？

GenBank,SwissProt 和 EMBL 文件中物种属性是一条注释。所有要做的工作就是检查序列注释信息看看物种属性是否符合要求。物种属性的名称依赖于数据源，在 EMBL,SwissProt 中用"OS"表示，在 GenBank 中用"Organism"表示。下面的程序从文件中读取序列并且根据物种属性过滤。你可以适当的更改源代码，从而可以应用到任何属性中去。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 读取 EMBL 文件
        BufferedReader br = new BufferedReader(new FileReader("F:/test/test"));
        SequenceIterator iter = SeqIOTools.readEmbl(br);

        // 读取过滤使用的物种名
        String species = "物种名";

        // 创建一个序列库来存储过滤的序列
        SequenceDB db = new HashSequenceDB();

        // 读取每条序列
        while(iter.hasNext())
        {
            Sequence seq = iter.nextSequence();
            Annotation anno = seq.getAnnotation();

            //检查注释中的物种属性
            if(anno.containsProperty("OS"))
            {
                String property = (String)anno.getProperty("OS");
                // 检查属性值，也可以使用正则表达式.将符合标准的序列加倒序列
库中。
                if(property.startsWith(species))
                {
                    db.addSequence(seq);
                }
            }
        }
        // 将经过筛选的序列库输出
        SeqIOTools.writeFasta(System.out, db);
---------------------------------------------------------------------------------------------------------------
```

3.6 位置和特征(location and feature)

3.6.1 如何定制一个点位置(PointLocaiton)?

在 Biojava 中，序列中的位置是用一个实现了位置(Location)接口的简单对象代表的。一个点位置是序列或者标志链(SymbolList)中单个标志(Symbol)的位置.点位置有公共的构造器可以很容易实例化。下面的程序展示了点位置的创建，定义了标志链中的单个标志。

注意 Biojava 使用的是生物学坐标，第一个位置是 1 而不是 0。

参考代码：

```
-----------------------------------------------------------------------------------------------------------------
// 创建第三个碱基为一个点位置
        PointLocation point = new PointLocation(5);

        // 打印位置
        System.out.println("Location: "+point.toString());

        // 创建一个标志链
        SymbolList sl = RNATools.createRNA("gcagcuaggcggaaggagc");
        System.out.println("SymbolList: "+sl.seqString());

        // 取得由位置定义的标志链（注意这里定义的是一个点位置，所以返回的是一个只有一个碱基的标志链）
        SymbolList sym = point.symbols(sl);
        System.out.println("Symbol specified by Location: "+sym.seqString());
-----------------------------------------------------------------------------------------------------------------
```

3.6.2 如何定制一个范围位置(RangeLocation)?

在 Biojava 中范围位置对象代表标志链中一段区域，由唯一的起始和终止来定义。

参考代码：

```
-----------------------------------------------------------------------------------------------------------------
// 创建一个第三个碱基到第八个碱基的范围位置
        Location loc = LocationTools.makeLocation(3,8);
        // 打印位置
        System.out.println("Location: "+loc.toString());

        // 创建一个标志链
        SymbolList sl = RNATools.createRNA("gcagcuaggcggaaggagc");
```

```
        System.out.println("SymbolList: "+sl.seqString());

        // 取得范围位置定义的标志链
        SymbolList sym = loc.symbols(sl);
        System.out.println("Symbols sepcified by Location: "+sym.seqString());
```

3.6.3 如何处理环状位置？

很多 DNA 分子,比如说质粒和细菌染色体是环状的。这样环状的分子上的位置就相对的模糊（比如说原点的定义不同，位置也不同）。在 Biojava 中真正的环状标志链并不存在。标志实际上是以数组或指针存储的。环状的效果可以用环状视图对象(CircularView)来模拟。这个对想要实现标志链视图(SymbolListView)。在标准的标志链中不可能访问超出标志链范围的位置。任何试图访问 0 位置或 length+1 位置都会跑出边界例外。但在环状视图中可以这么做，访问 0，-5 位置并且得到一个标志。在 CirclarView 中没有索引限制，索引 1 指向标志链中的第一个标志，索引 0 指向标记链中第一个标志前的标志，在这个例子中也就是标志链的最后一个标志。环状位置由环状位置类来处理。可以用位置工具(LocationTools)类来创建。下面的程序展示了这一点。

参考代码：

```
Location[] locs = new Location[3];
        // 分别创建一个 20 摩序列的第三碱基到第八碱基，第零碱基到第四碱基，第十八碱基到第二十四碱基的环状位置
        locs[0] = LocationTools.makeCircularLocation(3,8,20);
        locs[1] = LocationTools.makeCircularLocation(0,4,20);
        locs[2] = LocationTools.makeCircularLocation(18,24,20);
        for (int i =0; i < locs.length; i++)
        {
            // 打印位置
            System.out.println("Location: "+locs[i].toString());

            // 创建一个标志链
            SymbolList sl = DNATools.createDNA("gcagctaggcggaaggagct");
            System.out.println("SymbolList: "+sl.seqString());

            // 获得符合位置定义的序列部分
            SymbolList sym = locs[i].symbols(sl);
            System.out.println("Symbol specified by Location: "+sym.seqString());
```

```
        }
```

3.6.4 如何创建一个特征？

在 Biojava 中，特征（Feature）有点象具有位置（Location）属性的注释（Annotation）。

各种类型的特征必须都实现特征接口。所有的特征实现都包含一个叫做模版（Template）的内类。模版内类定义了创建特征必需的最小信息。当这个模版被当作参数传递给一个实现了特征拥有者（FeatureHolder）接口的实例的 createFeature()方法时，特征才被创建。

序列（Sequence）是特征拥有者（FeatureHolder）的子接口。所以序列能够拥有特征。注意：标志链（SymbolList）不能拥有特征。有趣的是，特征也是特征拥有者的子接口。所以特征能够拥有子特征形成一种嵌套层次。这种设计能够让基因特征拥有外显子特征，外显子特征拥有 snp 特征等等。特征拥有一个内置的安全机制能够避免自我包含。特征模版可以重新创建，也能从已有的特征中拷贝。

参考代码：

```
// 从链状特征中得到一个特征模版
        StrandedFeature.Template templ = new StrandedFeature.Template();
        // 填充模版
        templ.annotation = Annotation.EMPTY_ANNOTATION;
        templ.location = new RangeLocation(3,6);
        templ.strand = StrandedFeature.POSITIVE;
        templ.type = "interesting motif";
        // 拥有这个特征的序列
        Sequence seq = DNATools.createDNASequence("atgcgcttaag","seq1");
        System.out.println(seq.getName()+" contains "+seq.countFeatures()+" features");

        System.out.println("adding new feature...");

        // 创建一个序列特征
        Feature f = seq.createFeature(templ);
        System.out.println(seq.getName()+" contains "+seq.countFeatures()+" features");
```

3.6.5 如何根据类型来筛选特征？

如果你解析一个详细的 GenBank 文件，你将会面对包含不同类型的多个特征的序列。也许

你仅仅对其中一些类型的特征感兴趣，比如说"CDS"。你应该使用特征过滤器(FeatureFilter)来过滤这些特征，特征过滤器能产生一个特征拥有者(FeatureHolder)对象来存储你感兴趣的特征。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
Sequence seq = null;
//
        // 在这里编写初始化序列的代码，你可以读取象 GenBank 或者相似的文件生成带有多个不同特征的序列。
        //
        // 创建一个依据"CDS"类型的过滤器
        FeatureFilter ff = new FeatureFilter.ByType("CDS");

        // 获得筛选的特征
        FeatureHolder fh = seq.filter(ff);

        // 遍历特征拥有者(fh)所有的特征
        for (Iterator i = fh.features(); i.hasNext(); )
        {
            Feature f = (Feature)i.next();
        }
---------------------------------------------------------------------------------------------------------------
```

3.6.6 如何从序列中删除特征？

当你处理序列对象的时候可能希望能够删除一些特征。

参考代码：

```
---------------------------------------------------------------------------------------------------------------
// 读取 EMBL 文件
        BufferedReader br = new BufferedReader(new FileReader("F:/test/embl.txt"));
        SequenceIterator seqI = SeqIOTools.readEmbl(br);

        while(seqI.hasNext())
        {
            Sequence seq = seqI.nextSequence();
            // 获得所有在正链上的特征
            FeatureHolder fh = seq.filter(new FeatureFilter.StrandFilter(StrandedFeature.POSITIVE));

            // 遍历所有的特征
            for ( Iterator i = fh.features();i.hasNext();)
```

```
            {
                Feature ff=(Feature)i.next();
                // 删除每个特征
                seq.removeFeature(ff);
            }

            // 最后将编辑过的序列输出（EMBL format）
            SeqIOTools.writeEmbl(System.out, seq);
        }
```

3.7 BLAST 和 FASTA（待测试）

3.7.1 如何设置一个 BLAST 解析器解析 BLAST 结果？

一个生物信息学常遇到的任务是 BLAST 搜索。Biojava 能够解析哪些"BLAST"类型的输出，比如 Blast 或者 HMMER 的输出。Biojava 在这里使用了基于 SAX 事件处理的方法，通过设置监听器来解析这些输出。基本的流程如下：

Blast 输出 -> 生成 SAX 事件 -> 转化 SAX 事件 -> 创建结果对象 -> 将结果对象存储在链表中

输入流 -> BLAST 类型的 SAX 解析器(BLASTLikeSAXParser) -> 序列相似性适配器(SeqSimilartyAdapter) -> Blast 类型搜索创建器(BlastLikeSearchBuilder) -> 链表(List)

参考代码：

此程序作为参考，因为我找到的 blast 结果文件，他总是说不支持，尽管都是从 ncbi 上下载下来的，他还是说不支持，我不知道他到底支持什么版本的，所以这个是待测试的。

```
// 以流的方式得到 Blast 输出
            InputStream is = new FileInputStream("F:/test/blast.txt");
            // 创建一个 Blast 类型的 SAX 解析器
            BlastLikeSAXParser parser = new BlastLikeSAXParser();
            // 创建一个可以将事件传送给句柄的 SAX 事件适配器
            SeqSimilarityAdapter adapter = new SeqSimilarityAdapter();
            // 给 parser 设置 SAX 事件适配器
            parser.setContentHandler(adapter);
            // 保存序列相似性搜索结果(SeqSimilaritySearchResults)的链表
            List results = new ArrayList();
            // 创建在结果链表（Results List)中生成序列相似性搜索结果对象的搜索内容句柄(SearchContentHandler)
            SearchContentHandler builder = new BlastLikeSearchBuilder(results,
```

```
new DummySequenceDB("queries"), new DummySequenceDBInstallation());

// 解析文件，序列相似性搜索结果填充到结果链表
parser.parse(new InputSource(is));

// 输出一些 Blast 细节
for (Iterator i = results.iterator(); i.hasNext(); )
{
    SeqSimilaritySearchResult result = (SeqSimilaritySearchResult)i.next();
    Annotation anno = result.getAnnotation();

    for(Iterator j = anno.keys().iterator(); j.hasNext(); ){
    Object key = j.next();
    Object property = anno.getProperty(key);
    System.out.println(key+" : "+property);
}
System.out.println("Hits: ");

// 列出匹配
for (Iterator k = result.getHits().iterator(); k.hasNext(); )
{
    SeqSimilaritySearchHit hit = (SeqSimilaritySearchHit)k.next();
    System.out.println("\tmatch: "+hit.getSubjectID());
    System.out.println("\te score: "+hit.getEValue());
}

System.out.println("\n");
}
```

3.7.2 如何设置一个 FASTA 解析器解析 FASTA 搜索结果？

解析 FASTA 结果和解析 Blast 结果非常相似。仅仅将解析 Blast 结果的程序中的：

由于 blast 程序没有测试成功，所以这个也是待测试的。

XMLReader parser = new BlastLikeSAXParser();

替换成：

XMLReader parser = new FastaSearchSAXParser();

即可。

3.7.3 如何从搜索结果中提取信息？

Blast解析和Fasta解析程序已经讨论了一旦结果文件被解析就会生成一个存储序列相似性搜索结果对象的链表。每个查询（例如 Blast query）都对应着一个这样的链表。每个序列相似性搜索结果对象都包含着一个序列相似性匹配的链表存储具体的匹配。每个序列相似性搜索匹配对象都包含着一由序列相似性搜索子匹配组成的链表。这和 BLAST 报告的 HSP 相同。结果，匹配和子匹配包含着很多 getXXX()方法，能够提取存储的信息。

下面的小程序能够从 BLAST 或 FASTA 解析器返回得链表中提取匹配 ID（目标 ID),bit 分值，e 分值。

参考代码：

由于第一个 Blast 的程序没有调试成功，这里我只把这些的方法给出，但是都没有测试成功，待测试的。

```
---------------------------------------------------------------------------------------------------------------
private static void formatResults(List results)
{
        // 遍历所有序列相似性搜索结果
        for (Iterator i = results.iterator9); i.hasNext(); )
        {
                SeqSimilaritySearchResult result = (SeqSimilaritySearchResult)i.next();
                //遍历所有的匹配
                for (Iterator i2 = result.getHits().iterator(); i2.hasNext(); ){
                        SeqSimilaritySearchHit hit = (SeqSimilaritySearchHit)i2.next();
                        // 输出匹配 ID,bit 分值,e 分值
                        System.out.println("subject:\t"+hit.getSubjectID()   +"   bits:\t"+hit.getScore()+"
e:\t"+hit.getEValue());
                }
        }
}
---------------------------------------------------------------------------------------------------------------
```

3.8 计数和分布(count and distribution)

3.8.1 如何计算序列中的残基数？

计算序列中的残基数是一项相当普通的生物信息学任务。通常你可以构建一个整数数组使用一些索引系统。然而你可以使用 AlphabetIndex 做标准索引。

你可以从 AlphabetManager 中调用 getAlphabetIndex() 方法来得到一个字母索引

（AlphabetIndex）。因为这种任务很常用，所以 Biojava 将所有索引包装到一个索引计数（IndexCount）类中。这个类实现了计数（Count）接口。

下面的程序读取不同类型的序列文件对残基计数，将结果输出到标准输出。注意：这个程序不适合多义标志。如果想要计算多义标志则需要将多义标志的组成标志求和。参见解决方案二。

参考代码：

```
// 接受三个参数，第一个是文件名，第二个是序列类型（大小写敏感），第三个是字母表（大小写敏感）
```

解决方案 1

```
// 打开序列文件
        BufferedReader br = new BufferedReader(new FileReader(file));
        // 取得序列遍历器
        SequenceIterator iter =
        (SequenceIterator) SeqIOTools.fileToBiojava(type,alpa,br);
        // 对每个序列
        while(iter.hasNext())
        {
            Sequence seq = iter.nextSequence();
            // 如果需要的话，初始化计数
            if(counts == null)
            {
                counts = new IndexedCount((FiniteAlphabet)seq.getAlphabet());
            }
            // 遍历序列中的标志
            for (Iterator i = seq.iterator(); i.hasNext(); )
            {
                AtomicSymbol sym = (AtomicSymbol)i.next();
                counts.increaseCount(sym,1.0);
            }
        }
```

解决方案 2：

```
// 打开文件
        BufferedReader br = new BufferedReader(new FileReader(file));

        // 获得序列遍历器
        SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(type,alpa,br);

        //对每个序列
        while(iter.hasNext())
        {
```

```
            Sequence seq = iter.nextSequence();

            //必要的话,初始化计数
            if(counts == null)
            {
                counts = new IndexedCount((FiniteAlphabet)seq.getAlphabet());
            }

            // 遍历序列中的所有标志
            for(Iterator i = seq.iterator(); i.hasNext(); )
            {
                Symbol sym = (Symbol)i.next();

                // 如果是多义标志的话,将计数平分到组分标志中.例如标志 n 意味着这个位置四个碱基都有可能,所以一个 n 出现,则// 对每个碱基计数加 0.25.

                FiniteAlphabet subSymbol = (FiniteAlphabet)sym.getMatches();
                for(Iterator i2 = subSymbol.iterator(); i2.hasNext(); )
                {
                    AtomicSymbol sym2 = (AtomicSymbol)i2.next();
                    counts.increaseCount(sym2,1.0/(double)subSymbol.size());
                }
            }
        }
```

3.8.2 如何将计数转化为一个分布？

用分布工具类(DistributionTools)中的静态方法 countToDistribution()能很容易将计数转化为分布。

参考代码：

```
//创建一个计数
        Count c = new IndexedCount(alpha);
        c.increaseCount(RNATools.a(),35.0);
        c.increaseCount(RNATools.c(),44.0);
        c.increaseCount(RNATools.g(),68.0);
        c.increaseCount(RNATools.u(),34.0);

        System.out.println("COUNT");
        for (int i=0;i<alpha.size();i++)
        {
```

```
        AtomicSymbol s = (AtomicSymbol)index.symbolForIndex(i);
        System.out.println(s.getName()+" : "+c.getCount(s));
    }
    // 将计数转化为分布
    Distribution d = DistributionTools.countToDistribution(c);
    System.out.println("\nDISTRIBUTION");
    for(int i = 0; i<alpha.size();i++)
    {
        Symbol s = index.symbolForIndex(i);
        System.out.println(s.getName()+" : "+d.getWeight(s));
    }
```

3.8.3 如何从一个分布得到一条随机序列？

Biojava 中的分布对象有一个能随机化(Sampling)标志的方法。当你随机化标志后你能够创建一条随机序列。在分布工具类中提供一个静态方法 generateSequence()能实现上述任务。下面的程序使用了 DNA 字母表一致分布生成一条随机序列。虽然生成的序列每个碱基的分布都是 25%，但是序列各不相同。非一致分布能生成有偏好的序列。

参考代码：

```
// 创建一个一致分布
    Distribution dist = new UniformDistribution(DNATools.getDNA());
    // 生成 700bp 随机序列
    Sequence seq = DistributionTools.generateSequence("random seq",dist,700);
```

3.8.4 如何计算一个分布中的信息量或熵？

分布中的信息量或熵反映着分布的富集程度。可以用分布工具类中的静态方法计算香农信息(shannon information)和熵。香农信息的值反映着总的信息量，以 double 存储。熵值以 HashMap 存储每个标志和它对应着的熵。下面的程序对一个偏好分布的序列计算这两个值。

```
//创建一个偏好分布
    dist = DistributionFactory.DEFAULT.createDistribution(DNATools.getDNA());
    // 设置 a 的权重 0.97
    dist.setWeight(DNATools.a(),0.97);
    // 其余的为 0.1
```

```
            dist.setWeight(DNATools.c(),0.01);
            dist.setWeight(DNATools.g(),0.01);
            dist.setWeight(DNATools.t(),0.01);
// 计算信息量
        double info = DistributionTools.bitsOfInformation(dist);
        System.out.println("Information = "+info+" bits");
        System.out.print("\n");

        // 计算熵（以 log2 为底)
        HashMap entropy = DistributionTools.shannonEntropy(dist,2.0);
```

3.8.5 如何能找到一种简单的方法来判断两种分布是否具有相同的权重?

检测两个分布是否具有相同的矩阵是判别一个训练程序是否收敛或者两个序列是否来源于同一物种的好方法.当字母表很大时,对所有碱基作循环很麻烦.Biojava 的分布工具类提供了一个静态方法 areEmissionSpectraEqual()来为你完成上述任务.

参考代码：

```
FiniteAlphabet alpha = DNATools.getDNA();

        // 创建一个一致的分布
        Distribution uniform = new UniformDistribution(alpha);
//创建另一个分布, 权重是相同的
            Distribution dist = DistributionFactory.DEFAULT.createDistribution(alpha);
            dist.setWeight(DNATools.a(),0.25);
            dist.setWeight(DNATools.c(),0.25);
            dist.setWeight(DNATools.g(),0.25);
            dist.setWeight(DNATools.t(),0.25);

            // 检查两个分布是否一致
            boolean equal = DistributionTools.areEmissionSpectraEqual(uniform,dist);
```

3.8.6 如何对一个自定义的成分表创建一个 N 阶分布(order N distribution)?

这个例子创建了用户自定义的含有七个标志的字母表。用户自定义的标志和字母表能够用来创建标志链，序列，分布等等。当字母表管理器创建交叉产物字母表（CrossProductAlphabet)时,它将推断调节前的字母表(Conditioning alphabet)的阶为-1,调节好了的字母表(Conditioned alphabet)的阶为 1.

参考代码：

由于此代码太长，我这里就放上伪代码了，详细代码请参开代码文件。

```
// 创建一个“侏儒“ 字母表
/*...*/
//设置字母表的名字
/*...*/
// 将新创建的标志添加到字母表中
/*...*/
// 通常将新建的字母表注册到字母表管理器中
        /*...*/
        // 分布的阶数
        int order = 3;
        // 创建交叉产物字母表

        // 使用 n 阶分布工厂类创建分布

        //创建分布训练器

        //将分布注册到训练器

        //注释：我们能用"侏儒"字母表创建一条序列来检测我们创建的 n 阶分布.我们使用同一分布(能够随机化样本)将标志
        //添加到链表中.然后用这个链表构建标志链.

        //从"侏儒"字母表中随机创建一条标志链

        //创建一条标志链来检测分布

        //这个标志链被转化成一条 n 阶标志链从而能应用 n 阶分布.

        //转化成一条 n 阶标志链
```

```
//或者使用窗口标志链(WindowedSymbolList)
/
//将计数添加到分布中

// 不要忘记训练,否则没有权重会添加到分布中

//将分布写入 XML 文件中
XMLDistributionWriter writer = new XMLDistributionWriter();

writer.writeDistribution(ond, new FileOutputStream("dwarf.xml"));
```

3.8.7 如何将一个分布以 XML 格式输出?

如果你经常从大量的训练集中创建分布,并且希望存储这些分布待日后分析使用.一种可能的方法是将分布序列化以二进制存储.这种方法空间效率高,能被不同的 java 虚拟机上使用,但是这种方法也很容易出现由于 Biojava 版本的不同而导致的不兼容问题。而且,你有时候会希望能人为的读取分布.一个更好的方法是使用 XML 来存储分布.XML 是一种人和机器都能阅读的语言,独立于程序。下面的例子展示了如何将分布以 XML 格式输出,并读入 XML 格式的分布。

参考代码:

```
File temp = File.createTempFile("xmltemp", ".xml");

        //创建一个要输出的分布
        Distribution d = DistributionFactory.DEFAULT.createDistribution(DNATools.getDNA());

        //随机化这个分布
        DistributionTools.randomizeDistribution(d);

        //写到'temp'文件中
        DistributionTools.writeToXML(d, new FileOutputStream(temp));

        //再读回来
        Distribution d2 = DistributionTools.readFromXML(new FileInputStream(temp));

        //检查权重的一致性
        boolean b = DistributionTools.areEmissionSpectraEqual(d, d2);
```

3.9 权重矩阵和动态规划(weight matrix and dynamic programming)

3.9.1 如何使用一个权重矩阵来寻找一个模体？

一个权重矩阵能很好的展示一个比对结果或者是一个模体。他也能被用来作为一个打分矩阵在一条序列中搜索相似的模体。Biojava 的 org.biojava.bio.dp 包中包含了一个叫做权重矩阵的类。还有一个使用权重矩阵的权重矩阵注释器。

当搜索结果的分值超过一定的阈值时，能够往序列的任何部分添加特征。下面的程序从一个比对中获得一个权重矩阵，在用这个矩阵将超过 0.1 分值的结果作为特征注释这条序列。

参考代码：

```
// 进行一个模体的比对

        Map map = new HashMap();
        map.put("seq0", DNATools.createDNA("aggag"));
        map.put("seq1", DNATools.createDNA("aggaa"));
        map.put("seq2", DNATools.createDNA("aggag"));
        map.put("seq3", DNATools.createDNA("aagag"));
        Alignment align = new SimpleAlignment(map);

        // 创建一个模体分布数组

        Distribution[] dists = DistributionTools.distOverAlignment(align,false,0.01);

        // 创建一个权重矩阵

        WeightMatrix matrix = new SimpleWeightMatrix(dists);

        // 需要打分的序列

        Sequence seq = DNATools.createDNASequence("aaagcctaggaagaggagctgat","seq");

        // 用这个权重矩阵设定阈值 0.1 来注释这条序列
        WeightMatrixAnnotator wma = new WeightMatrixAnnotator(matrix,0.1);
        seq = wma.annotate(seq);
```

```
// 输出匹配信
```

3.9.2 如何创建一个隐马模型序列谱(Profile HMM)?(待测试)

隐马模型序列谱(例如 HMMER 程序使用的)是一种敏感度非常高的搜索模体的方法.通常使用 Baum-Welch 算法对一组序列进行训练,这组序列含有我们感兴趣模体.这个算法能将模型参数不断优化直到符合终止标准.一旦序列谱被构建,则能用 Viterbi 算法决定最可能生成观测序列的状态序列.如果在观测序列中有足够多的状态匹配或者使用一个定义了阈值的打分矩阵(比如说对数几率)且分值超过阈值则该序列被认为含有模体.

参考代码：

下边这个代码因为他也是只给一个思想，并没有一个完整代码，我只是猜测性的改正了一些错误，但是由于他的思想我还没有弄太清楚，所以完整代码没写出来，这个也是待测试。

```
// 创建一个十二列的 DNA 隐马模型序列谱,用默认的分布工厂类创建转移和生成分布
        ProfileHMM hmm = new ProfileHMM(DNATools.getDNA(),
        12,
        DistributionFactory.DEFAULT,
        DistributionFactory.DEFAULT,
        "my profilehmm");
        // 为模型创建一个动态规划矩阵
        DP dp = DPFactory.DEFAULT.createDP(hmm);

        //这时你要读取训练序列集.
        /*
         *
         * 编码内容
         *
         *
         * */

        // 存储训练集的数据库
        SequenceDB db = new HashSequenceDB();

        // 在这里编码,装载训练集
        /*
```

```
 *
 * 编码内容
 *
 * */

//现在,以一致的数值初始化所有的模型参数。另外，参数可以随机化或者猜测最符合模型的参数值。
//使用 Baum-Welch 算法优化序列。

// 将所有参数以同一数值训练
ModelTrainer mt = new SimpleModelTrainer();

// 注册模型
mt.registerModel(hmm);

// 使用空权重训练参数
mt.setNullModelWeight(1.0);
mt.train();

// 使用 BW 训练器，以动态规划矩阵为参数
BaumWelchTrainer bwt = new BaumWelchTrainer(dp);

// 使用匿名内类制定终止标准：20 遍
StoppingCriteria stopper = new StoppingCriteria()
{
    public boolean isTrainingComplete(TrainingAlgorithm ta)
    {
        return (ta.getCycle()>20);
    }
};

// 使用空模型（null model)权重 1.0 和上面定义的终止标准，优化动态规划矩阵
bwt.train(db,1.0,stopper);

//下面的例子对一条序列打分，并且输出状态序列
SymbolList test = null;
// 在这里编码，并且初始化待测序列
// 将待测序列装进一个数组中，这里使用数组是因为使用 HMM 的双序列比对需要在数组中的两条标志链。

SymbolList[] sla = {test};

// 解码最可能的状态序列，产生几率分值
StatePath path = dp.viterbi(sla,ScoreType.ODDS);
```

```
        System.out.println("Log Odds = "+path.getScore());
```

3.9.3 如何建立一个自定义的隐马模型(HMM)?

这个我参考的那个上边是以讲解 Biojava 源代码为例子的，由于对这个模型本身不是太了解，所以他的源码也没有看懂，这里不再给出，有兴趣可以参考：

http://www.cbi.pku.edu.cn/chinese/documents/PUMA/biojava/42.html

3.10 用户界面(user interfaces)

3.10.1 如何以树状结构展示特征和注释？

序列能够拥有键值对形式的注释和特征。并且特征能够拥有信息，注释，和嵌套的特征。嵌套的特征还能够拥有注释，更深层的嵌套特征。因此很适合用树状结构展示。

幸运的是，Biojava 团队创建了特征树（FeatureTree）类能够让你用树结构看特征。特征树继承了 JTree 类可以很容易地用在 GUI 中。树使用 SequenceDB 形式的数据，SequenceDB 能够从文本文件中读取。

下面的程序展示了如何使用特征树。他有两个参数。第一个是包含序列的文件名。第二个是代表数据格式的数字。

参考代码：

这个代码是图形界面的，但是基础知识呢，都在上边讲过，如果又不懂的，可参考上边的内容。

```
// 读取文本文件
        BufferedReader br = new BufferedReader(new FileReader("F:/test/embl.txt"));

        // 获取文件类型
        String format = "EMBL";

        // 获取字母表
        String alpha = "DNA";

        //将序列读入序列库，这个序列库将当作树的数据模型
        SequenceDB db = new HashSequenceDB();
        SequenceIterator iter = (SequenceIterator)SeqIOTools.fileToBiojava(format,alpha,br);
```

```
while(iter.hasNext())
{
    db.addSequence(iter.nextSequence());
}
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
TreeFrame treeFrame = new TreeFrame();
// 设置序列库作为树的数据模型
treeFrame.getFeatureTree().setSequenceDB(db);
treeFrame.pack();
treeFrame.show();
```

3.10.2 如何图形化一条序列?

当创建生物信息学 GUI（图形化用户接口）时，我们希望能够在 GUI 中显示序列。Biojava 中包含很多 GUI 组件能绘制序列的各个部分。最基本的序列绘制组件是序列绘制上下文（SequenceRenderContext)。他负责存储序列并指导序列绘制器绘制序列。在 Biojava 中有很多序列绘制器的实现。标志序列绘制器（SymbolSequenceRender)能依次展现序列。下面的程序展示了如何使用序列绘制上下文和序列绘制器绘制序列。

参考代码：

```
this.getContentPane().setLayout(new BorderLayout());
        this.getContentPane().add(jPanel,BorderLayout.CENTER);
        this.setTitle("seqView");
        jPanel.add(seqPanel, BorderLayout.CENTER);

        // 设置要显示的序列
        seqPanel.setSequence(seq);

        // 设置负责绘制的对象
        seqPanel.setRenderer(symSeqRenderer);

        // 要显示的序列的范围
        seqPanel.setRange(new RangeLocation(1,seq.length()));
```

3.10.3 如何展示序列标尺？

当你图形化序列时需要展示标尺，这样你就很容易定位碱基。Biojava 包含一个叫标尺绘制

器（RulerRenderer)的序列绘制器实现能够绘制序列标尺。因为序列绘制上下文(SequenceRenderContext)同一时间只能使用一个序列绘制器，所以你必须使用多行绘制器(MutiLineRenderer)。多行绘制器实现了序列绘制器接口，能够包装多个序列绘制器，在多个轨道上调节他们的图形。

下面的程序使用了标尺绘制器，多行绘制器。

参考代码：

代码没有什么要讲的，都是设置 GUI 的，因此可参考详细源代码。

3.10.4 如何图形化一个特征？

特征由实现特征绘制器(FeatureRenderer)接口的类负责绘制。特征绘制器就象序列绘制器那样工作负责绘制序列中的特征。一个序列绘制上下文不能直接和特征绘制器值作用，所以使用特征块序列绘制器(FeatureBlockSequenceRenderer)当作中间的代理，将特征绘制器包装起来。

下面的程序使用了特征块序列绘制器和特征绘制器。

参考代码：

解释同上

3.11 如何设置 BioSQL?(待测试)

前提：这个是拷贝过来的，我自己还没有测试。

本文档描述如何安装使用 BioSQL。BioSQL 是 OBDA 标准的一部分，作为支持多种语言的通用序列数据库模式由 Open Bioinformatics Foundation 开发。

BioSQL 在设计上采用中立立场。我基于我对它的认识编写了这个指南。我使用的是 RedHat7.2 系统的 x86 机器。

安装 Postgresql

如果你没有安装的话，Postgresql 可以用 rpm 安装：

rpm -ivh postgresql-7.2.1-5.i386.rpm postgresql-libs-7.2.1-5.i386.rpm

postgresql-server-7.2.1-5.i386.rpm

通常需要 Root 权限（如果不需要的话，你的机器会有安全问题！）。你还需要 JDBC 使得 Java 程序可以连接到你的 PostgreSQL 数据库上。可以通过安装 postgresql-jdbc-7.1.3-2.i386.rpm 来实现。不过我建议你从这里下载安装最新版本。你还需要将实现 JDBC 的 jar 文件添加到你的 CLASSPATH 中。安装时一个叫 postgresql 的控制脚本会在/etc/init.d 中。当这个脚本第一次运行时，将会创建并初始化一个数据库簇（database cluster)。这个簇被 PostgreSQL 用来存储数据文件。

在 RedHat7.2 中，默认的数据库簇安装目录在/var/lib/pgsql/下。这可能有缺点，因为通常这个/var 目录很小。在这一步，你可以用连接将这个目录指向其他一个空间足够大的目录。我强烈建议你现在就这么做。

在这一步中，你将创建一个数据库和用户名。我不建议你使用超级用户 postgres 取做很多工作，这个超级用户应该拿来专门做一些管理工作而不是应用工作。讲一些题外话，PostgreSQL 的认证将会影响到你能够做什么事情。PostgreSQL 有很多途径能实现这点。默认的安装只允许被本地用户连接，只允许和本地用户同名的用户连接数据库。这对于练习数据库来书足够，但对于应用，比如说你想建立 BioSQL 数据库，并允许很多本地用户甚至是远程用户连接它，这样的话这个默认认证远远不够。

PostgreSQL 在它的文档中阐述了很多机制原理。认证机制在这里阐述。你可以考虑使用密码认证，当你希望认证远程用户时，请使用 md5 加密选项。在 RedHat7.2 安装中，你需要编辑/var/lib/pgsql/data/pg_hba.conf 文件实现上述选项。这个文件的位置根据不同的安装版本而不同。在 RedHat7.2 初装时，PostgreSQL 需要 Root 权限去进一步安装。这时不能以超级用户 postgres 登录，你可以从 Root 用户中执行：

创建数据库

$ su postgres -c 'createdb <insert db name here>'

创建用户

$ su postgres -c 'createuser <insert user name here>'

在本指南中，我将不会改变认证，这样数据库的名字将和你的用户名一致。用户名为 gadfly，这将在数据库和用户的名字体现。另外一个变化是启动 TCP/IP 连接，默认安装受 Unix 域套接字限制是不允许 TCP/IP 连接的。如果不启动这个选项，PostgreSQL JDBC 将不能使用。要开启 TCP/IP 选项，在启动脚本中加上"-i"选项。编辑/etc/init.d/postgresql 作如下改变：

su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -D $PGDATA -p /usr/bin/postmaster start > /dev/null

2>&1" < /dev/null

改成：

su -l postgres -s /bin/sh -c "/usr/bin/pg_ctl -o "-i" -D $PGDATA -p /usr/bin/postmaster start > /dev/null 2>&1" < /dev/null

同时/var/lib/pgsql/data/pg_hda.conf 文件也需要改写，允许 TCP/IP 连接。将：

#host all 127.0.0.1 255.255.255.255 trust

这行解注释。

这几个操作都需要 Root 权限。查看建议，这样能使你拥有安全的系统环境。

在 RedHat7.3 中 postgresql 的 pgsql 语言没有被激活。BioSQL 使用他来提高效率。你需要激活他：

su postgres -c 'createlang plpgsql template1'

安装 BioSQL

首先，PostgreSQL 必须启动。查看数据库状态：

$ /etc/rc.d/postgresql status

如果没有启动：

$ /etc/rc.d/postgresql start

启动数据库。必须以 Root 权限完成上述操作。使用 SysV 初始化系统启动 PostgreSQL 在系统启动时。

你需要三个脚本初始化新的 BioSQL 数据库:

biosql-accelerators-pg.sql

biosqldb-assembly-pg.sql

biosqldb-pg.sql

从这里下载。

我们需要将模式添加到我们创建的数据库中。如下：

$ psql gadfly

Welcome to psql, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms

\h for help with SQL commands

\? for help on internal slash commands

\g or terminate with semicolon to execute query

\q to quit

gadfly=> \i biosqldb-pg.sql

CREATE

psql:biosqldb-pg.sql:13: NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index 'biodatabase_pkey' for table 'biodatabase'

CREATE

<rest of output snipped>

INSERT 16862 1

psql:biosqldb-pg.sql:304: NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index 'cache_corba_support_pkey' for table 'cache_corba_support'

CREATE

gadfly=> \i biosqldb-assembly-pg.sql

<rest of output snipped>

gadfly=> \i biosql-accelerators-pg.sql

<rest of output snipped>

gadfly=> \q

$

我们看看上述过程。psql 是 PostgreSQL 的交互 Shell，我们用它来连接到 PostgreSQL 服务器上。连接到我们先前创建的名字叫 gadfly 数据库上。psql 启动并显示用户提示符。所有的 psql 命令以一个反斜杠开始(\)。\i 意味着 psql 从一个文件中读取命令。我们让 psql 从 biosqldb-pg.sql, biosqldb-assembly-pg.sql and biosql-accelerators-pg.sql 中读取命令。psql 根据这些文件创建 BioSQL 数据库模式，输出他的处理过程。最后用\q 命令退出 psql。至此，BioSQL 数据库创建完成。

务必记住如果你在你的程序中不显式的装载 JDBC 驱动，你应该设置 Java 环境变量告诉虚拟机在哪里装载驱动。例如：java -Djdbc.drivers=org.postgresql.Driver <whatever your java code is>

注意： 如果你使用的是新加坡模式的 Biojava1.3，不要安装 biosqldb-assembly-pg.sql，biosql-accelerators-pg.sql ，你只需要新版本的 biosqldb-pg.sql。如果你安装了其它的模式文件可能会有性能问题。这个提示将会根据建议不断更新。

4. 总结

以上只是 Biojava 的部分功能以及代码，他的强大之处并不止这些，因此，欢迎大家能够提出更多的功能使用方法，很多东西我也是一知半解，并没有完全掌握，如果有什么问题，可以找我（胡始昌）讨论。另外，希望这些东西能够给大家带来帮助。^_^