Jcseg 开源中文分词器开发说明文档

(注：该文档只适用于 **1.8.8** 及其以后的版本)

一. 关于 *jcseg:*

jcseg 是使用 Java 开发的一款开源的中文分词器, 使用 mmseg 算法. 分词准确率高达 98.4%, 支持中文人名识别, 同义词匹配, 停止词过滤..., 详情请查看 jcseg 官方首页.

官方首页: https://code.google.com/p/jcseg/

下载地址: https://code.google.com/p/jcseg/downloads/list

二. 安装 *jcseg:*

先到 jcseg 官方地址下载最新版本的 jcseg: jcseg-{version}-src-jar-dict.zip, {version}表示版本号, 下同.

解压 jcseg-{version}-src-jar-dict.zip 到你要安装的目录即可. 例如: Linux: /java/jcseg, WinNT: D:/jcseg

三. 配置 *jcseg:*

jcseg 词库的加载以及很多功能的开启和关闭都可以在 jcseg.properties 中配置.

1. 基本配置：

(1).让 jcseg 找到 jcseg.properties 配置文件:

jcseg.properties 配置文件查找方式:

1). jar home 路径搜索. 2). classpath 中搜索. 3). user home 路径搜索.

jcseg.properties 配置文件查找顺序: jcseg 首先会在 jar home 路径下搜索 jcseg.properties 配置文件, 如果没有找到, 接下来会在 classpath 中搜索(1.8.3 版中我已将一份全部为默认选项的 jcseg.properties 文件放入了 jcseg-core-{version}.jar 中), 所以也就不会出现配置文件查找不到的情况了, 另外当前版本的 jcseg 还支持在 user home 下查找 jcseg.properties 文件(System.getProperty("user.home")来获取 user home 路径).

对于使用默认选项的用户来说, 就不需要这个文件了, 如果需要自主配置该文档的, 将更改后的 jcseg.properties 文件放在 jar home 即可.

当然你也可以把更改后的 jcseg.properties 文件重新打包到 jcseg-core-{version}.jar 中去, 基于需要修改的方便性, 通常放于 jar home 下, 来覆盖 jcseg-core-{version}.jar 中的默认配置.

4). 自定义配置文件查找方式：从 **1.8.8** 版本开始 **jcseg** 支持自定义配置文件查找方式, 有利于复杂项目的配置文件的统一管理. 请看下面的 **Jcseg** 二次开发.

(2). 让jcseg找到词库:

jcseg找到jcseg.properties配置文件后, jcseg.properties中 lexicon.path 的值就是词库文件的绝对路径.
例如, 你的jcseg安装在D:/jcseg中, 则:
设置jcseg.properties中的lexicon.path=D:/jcseg/lexicon
此时, 词库文件的绝对路径为: lexicon.path, 即: D:/jcseg/lexicon

lexicon.path默认值为{jar.dir}/lexicon表示jar home/lexicon目录下, 建议词库和jar文件分开存放, 然后配置lexicon.path来指向词库.

注: 从 jcseg 1.8.8 开始, 去掉了 jcseg.properties 中的 lexicon.dir 选项. 并且支持自定义词库目录全部词库文件词条或者指定词库文件词条加载, 方便词库管理的二次开发. 请看下面的 Jcseg 二次开发.

提示: 词库不需要压缩到jcseg jar包中去, 也不建议这么做, 这样方便对词库进行随时的维护, 词库可以放在任何地方, 只要在jcseg.properties中配置lexicon.path来指向词库目录即可.

(3). 更多配置:
你可以更改最大化匹配长度, 关闭或者开启中文人名识别, 关闭或者开启切分同义词追加, 关闭或者开启停止词过滤功能...,
请参考下面的jcseg.properties配置项以及相关含义.

修改后, 重启jcseg相关服务即可.

2. jcseg.properties 配置项以及含义: (可以先不用理会)

```
# jcseg function
#-正向最大化匹配数目(建议位于4-7之间)。
jcseg.maxlen=1

#-开启中文人名识别(1.7.0后, 0关闭, 1开启)。
jcseg.icnname=1

#-中英混合词最大中文词数，例如：A计划 A后面有两个字“计划”。
jcseg.mixcnlen=2

#最大的配对标点内容长度。
jcseg.pptmaxlen=15

#-姓氏修饰词长度，例如：老陈 中的“陈”(通常为1)。
jcseg.cnmaxlnadron=1

#是否自动过滤停止词(0关闭, 1开启)
jcseg.clearstopword=1

#是否自动中文数字转阿拉伯数字(0关闭, 1开启)
jcseg.cnnumtoarabic=1

#是否自动中文分数转阿拉伯分数(0关闭, 1开启)
jcseg.cnfratoarabic=1

#-姓名成词歧义阈值(不用更改, 除非你知道你改了什么)。
jcseg.nsthreshold=1000000
```

```
# about the lexicon
#-词库文件前缀(例如: lex-main.lex)。
lexicon.prefix=lex

#-词库文件后缀(例如: lex-main.lex)。
lexicon.suffix=lex

#-词库存放路径({jar.dir}/lexicon 为默认路径, 表示词库位于 jar 目录下的 lexicon 目录下)
lexicon.path={jar.dir}/lexicon

#-是否词库更新自动加载(1 开启, 0 关闭)
lexicon.autoload=1

#-词库更新轮询时间(单位: 秒)
lexicon.polltime=120

# lexicon load
#载入词库时是否载入词条的词性。(0 关闭, 1 开启)
jcseg.loadpos=0

#载入词库时是否载入词条的拼音(1.7.0 后, 0 关闭, 1 开启)。
jcseg.loadpinyin=1

#载入词库时是否载入词条的同义词(0 关闭, 1 开启)。
jcseg.loadsyn=1
```

四. 运行 *jcseg* 测试程序:

配置好后, 相信你想知道配置是否正确, 或者说想试试 jcseg 的分词效果:
切换到 jcseg 安装目录, 然后运行:

```
java -jar jcseg-core-{version}.jar
```

你会看到各个配置项的当前值, 以及默认的分词效果, 和如下提示界面:

```
+--------jcseg chinese word segment demo-------+
|- Suggest email: chenxin619315@gmail.com  |
|- Run :quit or :exit to exit.                                  |
+---------------------------------------------------------------+
jcseg>>
```

在 jcseg>>提示符后输入你要分词的内容, 按 enter 键即可.

例如:

```
jcseg>> jcseg 是使用java 开发的一款开源中文分词组件.
分词结果:
jcseg 使用 java 开发 一款 开源 中文分词 组件
Done, total:26, split:8, cost: 0.00000sec
```

total 为总字符数, split 为切分之后的词条数, cost 为耗费的时间.

五. *lucene* 分词用法:

```
//导入 jcseg-core-{version}.jar 和 jcseg-analyzer-{version}.jar

/**
1.JcsegTaskConfig.COMPLEX_MODE 为复杂模式:
特点:四种过滤算法。*/

/**
2.JcsegTaskConfig.SIMPLE_MODE 为简易模式
特点:只使用了最大化过滤算法,其他的同复杂模式。
*/
Analyzer analyzer = new JcsegAnalyzer4X(JcsegTaskConfig.COMPLEX_MODE);

//非必须(用于修改默认配置): 获取分词任务配置实例
JcsegAnalyzer4X jcseg = (JcsegAnalyzer4X) analyzer;
JcsegTaskConfig config = jcseg.getTaskConfig();

//追加同义词到分词结果中, 需要在 jcseg.properties 中配置 jcseg.loadsyn=1
config.setAppendCJKSyn(true);
//追加拼音到分词结果中, 需要在 jcseg.properties 中配置 jcseg.loadpinyin=1
config.setAppendCJKPinyin();
//更多配置, 请查看 com.webssky.jcseg.core.JcsegTaskConfig 类
```

接下来把 analyzer 给 lucene 即可.

六. *solr* 分词用法:

1. 从 jcseg 附件解压目录中将 jcseg-core-{version}.jar 和 jcseg-solr-{version}.jar 复制到 solr 的类库目录中.

2. 在 solr 的 scheme.xml 加入如下两种配置之一:

```
    <!----复杂模式分词: -->
    <fieldtype name="textComplex" class="solr.TextField">
        <analyzer>
            <tokenizer class="com.webssky.jcseg.solr.JcsegTokenizerFactory"
mode="complex"/>
        </analyzer>
    </fieldtype>

    <!----简易模式分词: -->
    <fieldtype name="textSimple" class="solr.TextField">
        <analyzer>
            <tokenizer class="com.webssky.jcseg.solr.JcsegTokenizerFactory"
mode="simple"/>
        </analyzer>
    </fieldtype>
```

如果需要做类似上面 lucene 的配置, 需要更改 com.webssky.jcseg.solr.JcsegTokenizerFactory 分词工厂.

七. *Jcseg* 二次开发*(Jcseg API):*

你需要一个 JcsegTaskConfig 配置实例, 一个 ADictionary 词库实例和一个 ASegment 分词实例来完成分词工作, 三个实例的创建方法如下:
(完整的例子可以查看 com.webssky.jcseg.test.JcsegTest.java)

1. 创建 JcsegTaskConfig 分词配置实例:

JcsegTaskConfig 构造方法:

JcsegTaskConfig()
JcsegTaskConfig(java.lang.String proFile)

(1). 创建从默认的 jcseg.properties 中初始化的 JcsegTaskConfig 对象:

```
//该方法会自动查找 jcseg.properties 配置文件
//然后依据 jcseg.properties 中的选项初始化 JcsegTaskConfig.
JcsegTaskConfig config = new JcsegTaskConfig();

//或者使用如下方式(指定自定义配置文件路径为 null).
JcsegTaskConfig config = new JcsegTaskConfig(null);
```

(2). 创建从指定的 jcseg.properties 中初始化的 JcsegTaskConfig 对象:

```
//依据 proFile 指定的配置文件创建 JcsegTaskConfig.
JcsegTaskConfig config =
      new JcsegTaskConfig("/java/jcseg/jcseg.properties");
```

(3). 从给定的jcseg.properties配置文件中重置JcsegTaskConfig:

```
//从指定的jcseg.properties配置文件中重置JcsegTaskConfig选项
config.resetFromPropertyFile("/java/jcseg/jcseg.properties");
```

2. 创建 **ADictionary** 分词词库实例**:**

使用 com.webssky.jcseg.core.DictionaryFactory 创建 ADictionary 实例:
注: jcseg 默认 ADictionary 实现为: com.webssky.jcseg.Dictionary, 以下方法都是创建该默认实现的实例.
构造方法如下:

```
Adictionary(JcsegTaskConfig config, java.lang.Boolean sync)
```

参数说明:
config: 分词配置对象, 里面有词库加载所需要的信息.
sync: 是否创建同步词库. true 表示创建同步词库,false 表示创建非同步词库.

注: 通常都是创建非同步词库(比同步词库快),如果在 **jcseg** 运行的过程中,有程序会调用 **jcseg** 的词库 **API** 来增加或者删除词库, 这个情况下你就需要创建同步词库, 例如: 如果开启了 **jcseg** 的词库更新自动加载功能, 就需要创建同步词库(通过 **DictionaryFactory** 来创建, **jcseg** 自己处理好了).

为了兼容或者说方便扩展 jcseg 不是直接 new 一个 ADictionary(其实现类)实例, 而是通过 DictionaryFactory 来创建词库对象.

(1). 依据给定的JcsegTaskConfig 创建 ADictionary:

```
//config为上面创建的JcsegTaskConfig对象.
//如果给定的JcsegTaskConfig里面的词库路径信息正确(是从
jcseg.properties中初始化的就对了),
//Adictionary会依据JcsegTaskConfig里面的词库信息加载全部有效的词库.
//并且该方法会依据config.isAutoload()来决定词库的同步性还是非同步性.
//config.isAutoload()为true就创建同步词库, 反之就创建非同步词库.
//config.isAutoload()对应jcseg.properties中的lexicon.autoload
ADictionary dic = DictionaryFactory
                .createDefaultDictionary(config);
```

(2). 依据给定的JcsegTaskConfig 和 sync 创建 ADictionary:

```
//创建一个非同步的ADictioanry.
ADictionary dic = DictionaryFactory
                .createDefaultDictionary(config, false);

//创建一个同步的ADictioanry.
ADictionary dic = DictionaryFactory
                .createDefaultDictionary(config, true);

//依据config.isAutoload()来决定同步性
ADictionary dic = DictionaryFactory
```

```
            .createDefaultDictionary(config, config.isAutoload());
```

(3). ADictionary 词库加载 API:

```
//指定 ADictionary 加载给定目录下的所有词库文件的词条.
//config.getLexiconPath 为词库文件存放有效目录.
dic.loadFromLexiconDirectory(config.getLexiconPath());

//指定 ADictionary 加载给定词库的词条.
//config 为上面创建的 JcsegTaskConfig 实例.
dic.loadFromLexiconFile(config, "/java/lex-main.lex");
```

注：加载给定目录下的所有词库文件词条或者加载给定词库文件的词条的 API 为 Jcseg 词库更新自动加载提供了很大的方便性.

3. 创建 ASegment 或者 ISegment 分词实例：

Jcseg 的主分词类程序在 com.webssky.jcseg.ASegment 抽象类中得到具体实现，该类实现了 Jcseg 主 API 接口 com.webssky.jcseg.core.ISegment.
ComplexSeg 继承 ASegment 实现了复杂模式分词，SimpleSeg 继承了 ASegment 实现了简易模式分词.

Asegment 构造方法:

Asegment(JcsegTaskConfig config, Adictionary dic)
Asegment(Reader input, JcsegTaskConfig config, Adictionary dic)

注意上面的 Input 的 Reader 是 java.io 包中的 Reader.

参数说明:
config: 分词配置实例，参考上面的创建 JcsegTaskConfig.
dic: 词库实例， 参考上面的创建 ADictionary.

com.webssky.jcseg.test.JcsegTest 有详细的测试程序.

常用创建 ISegment 或者 ASegment 实例方法：

(1). 不带 *Input*(构造方法一)：

```
//创建 JcsegTaskConfig 分词任务实例
//即从 jcseg.properties 配置文件中初始化的配置
JcsegTaskConfig config = new JcsegTaskConfig();

//创建默认词库(即: com.webssky.jcseg.Dictionary 对象)
//并且依据给定的 JcsegTaskConfig 配置实例自主完成词库的加载
ADictionary dic = DictionaryFactory
                .createDefaultDictionary(config);
```

```
//依据给定的ADictionary和JcsegTaskConfig来创建ISegment
//通常使用SegmentFactory#createJcseg来创建ISegment对象
//将config和dic组成一个Object数组给SegmentFactory.createJcseg方法
//JcsegTaskConfig.COMPLEX_MODE表示创建ComplexSeg复杂ISegment分词对象
//JcsegTaskConfig.SIMPLE_MODE表示创建SimpleSeg简易Isegmengt分词对象.
ASegment seg = SegmentFactory
            .createJcseg(JcsegTaskConfig.COMPLEX_MODE,
                    new Object[]{config, dic});

//设置要分词的内容
String str = "研究生命起源。";
seg.reset(new StringReader(str));

//获取分词结果
IWord word = null;
while ( (word = seg.next()) != null ) {
      System.out.println(word.getValue());
}
```

可以反复利用创建的 ISegment 对象, 通过 ISegment.reset(java.io.Reader)来重置分词内容即可.

(2). 带 input: 也就是第二个构造方法, 和第一个的不同点在于, 同时把要分词内容的流对象也传递进去了:

```
//创建 JcsegTaskConfig 分词任务实例
//即从 jcseg.properties 配置文件中初始化的配置
JcsegTaskConfig config = new JcsegTaskConfig();

//创建默认词库(即: com.webssky.jcseg.Dictionary 对象)
ADictionary dic = DictionaryFactory.createDefaultDictionary();
//依据 JcsegTaskConfig 配置中的信息加载全部 jcseg 词库.
dic.loadFromLexiconDirectory(config, config.getLexiconPath());

//要被分词的文本
String str = "研究生命起源。";

//依据给定的分词流对象, ADictionary和JcsegTaskConfig来创建ISegment
//通常使用SegmentFactory来创建ISegment对象
//将config和dic组成一个Object数组给SegmentFactory.createJcseg方法
//JcsegTaskConfig.COMPLEX_MODE表示创建ComplexSeg复杂ISegment分词对象
//JcsegTaskConfig.SIMPLE_MODE表示创建SimpleSeg简易Isegmengt分词对象.
ASegment seg = SegmentFactory
          .createJcseg(JcsegTaskConfig.COMPLEX_MODE,
             new Object[]{new StringReader(str), config, dic});

//获取分词结果
IWord word = null;
while ( (word = seg.next()) != null ) {
      System.out.println(word.getValue());
}
```

此处同样可以反复利用创建的 ISegment 对象, 通过 ISegment.reset(java.io.Reader)来重新设置分词内容即可.
更多例子请参考 jcseg 分词测试程序: com.webssky.jcseg.test.JcsegTest 类源码

注意: 通过此方法创建的 ISegment 对象是非线程安全的.

jcseg 目前创建的分词实例都是非线程安全的, 但是这并不影响你的使用, 不同的线程创建不同的 ISegment 分词对象即可, 同时共用词库实例(lucene 和 solr 内部接口就是这样的). 如果对一个 ISegment 对象多线程调用就不能保证分词的顺序, 要实现多线程调用一个 ISegment 分词对象还需要做一个中间存储过程.
如果你想实现多线程也可以:
你可以把要分词的流拆分下, 然后创建多个 ISegment 对象在不同的线程下工作, 每个 ISegment 负责对一个拆分得到的流进行切分. 这多个 ISegment 分词对象共用同一个词库.
jcseg 后续会自带这种功能, 目前需要开发者自己开发.

Jcseg 官网中有一个 issule 讨论到了这个问题:
https://code.google.com/p/jcseg/issues/detail?id=9#c3

八. *jcseg* 词库管理:

因为文档可能会更改, 请查看 jcseg 的 Wiki 文档: google code 一向受到阻碍, 打开有时候很慢, 耐心点吧.

1.jcseg 词库结构: https://code.google.com/p/jcseg/wiki/JcsegLexicon
2.给 jcseg 添加新词库或者新词条:
https://code.google.com/p/jcseg/wiki/WikiJcseg
3.jcseg 词库类别以及说明:
https://code.google.com/p/jcseg/wiki/JcsegLexiconType

注意: 当你更新了词库后, 如果你需要 jcseg 马上响应你对词库的更改, 你需要重启 jcseg 相关服务, 也就是让 jcseg 重新加载词库.
当然, jcseg 提供了 API 来操作内存中的词库, 很多网友问道, 更改了 jcseg 的词库后, 怎么让 jcseg 立即响应更改来使用更新后的词库, 这个需要你自己调用 jcseg 的 API 来完成相关服务, 当你更改词库后, 调用你的程序来重新加载词库即可.
1.8.8 开始 jcseg 支持词库更新自动加载, 请往下看 ^_^.

Jcseg 词库 API:

jcseg 词库 API 定义在 com.webssky.jcseg.ADictionary 抽象类中. 提供了方法来添加/编辑/删除/查找/批量加载 词条.

1. jcseg 词库类别以及关键字:

jcseg 对词库进行了分类, 每个词库文件的第一行都有一个关键字用于对词库分类, 请不要删除了(删除了就是默认类别:CJK_WORDS).
以下是词库类别的以及含义说明:

```
CJK_WORDS
#含义: 普通 CJK 词库, 中文,日文,韩文普通词库.

CJK_UNITS
```

```
    #含义: CJK 字符单位词库, 例如: 斤, 公里

    EC_MIXED_WORD
    #含义: 英文字符和 CJK 字符混合词库, 例如: B 超, x 射线

    CE_MIXED_WORD
    #含义: CJK 字符和英文字符混合词库, 例如: 卡拉 ok, 漂亮 mm

    CN_LNAME
    #含义: 中文姓氏词库, 例如: 陈, 阳, 罗

    CN_SNAME
    #含义: 中文姓名"单字姓名"名词库, 例如: 张三中的"三", 李四中的"四"

    CN_DNAME_1
    #含义: 中文双子姓名, 首字词库, 例如: 李大头中的"大"

    CN_DNAME_2
    #含义: 中文双字姓名中的, 尾字词库, 例如: 李大头中的"头"

    CN_LNAME_ADORN
    #含义: 中文姓氏修饰词, 例如: 老陈中的"老", 小李中的"小", jcseg 会自动识别"老陈", "小陈"这类词

    EN_PUN_WORDS
    #含义: 英文和标点组成词, 例如: c++, c#, g++

    STOP_WORD
    #含义: 所有停止词词库
```

2. 获取正在工作的 **ADictionary** 实例**:**

通常用的是 com.webssky.jcseg.Dictionary 默认词库实现. (意味着你也可以使用自己的词库实现, 继承 ADictionary 即可)

给 lucene 的 JcsegAnalyzer4X 和给 solr 的 JcsegTokenizerFactory 都可以通过对应对象的 getDict()方法来获取正在使用的词库实例. 设获取的词库对象为 dic.

3. 添加新词条(注: 此添加只是在内存中, 并不会应用更改到词库文件中):

API:

```
void add(int t, String key, int type)
```

说明:

t : 词库的类别

key: 词条

type: 类别, 使用 IWord.*T_CJK_WORD* 即可

例如: 添加新词条"研究"到 CJK 主词库中:

```
dic.add(ILexicon.CJK_WORDS, "研究", Iword.T_CJK_WORD);
```

4. 删除给定词条**:**

API:

```
void remove(int t, String key)
```

说明:

t : 词库类别

key: 要被删除的词条
例如: 删除停止词词库中的”我们”:

```
dic.remove(ILexicon.STOP_WORD, “我们”);
```

5. 查询词条(包括给词条添加同义词和拼音)
API:

```
IWord get(int t, String key)
```

说明:
t : 词库类别
key: 要查找的词条:
返回值: 成功返回 IWord 对象, 失败返回 null
例如: 在 CJK 主词库中查找”研究”:

```
IWord w = dic.get(ILexicon.CJK_WORDS, “研究”);
```

6 编辑给定的词条(添加同义词, 修改拼音, 添加词性, 修改语素自由度等等)
通过上步返回的 w 对象再调用 IWord 的 API 就可以实现对应的功能了:
(1).添加同义词: w.addSyn(“研发”);
(2).添加词性: w.addPartSpeech(“n”);
(3).设置词条拼音:w.setPinyin(“yan fa”);
更多方法请参考 com.webssky.jcseg.core.IWord 接口或者 com.webssky.jcseg.Word 类

7. 依据给定的 **config** 和词库目录加载该目录下所有词库词条:

```
dic.loadFromLexiconDirectory(config, config.getLexiconPath());
```

注: 之所以需要 config 是因为 config 里面有来此 jcseg.properties 中的词库文件的前缀和后缀还有词库路径等信息.

8. 依据给定的 **config** 和词库文件加载该词库文件中的词条:

```
dic.loadFromLexiconFile(config, "/java/lex-main.lex");
```

注: 给定的词库必须为 jcseg 标准词库, 请查看 jcseg 词库结构:
https://code.google.com/p/jcseg/wiki/JcsegLexicon.

New!!!

如果你是直接更改了词库文件或者新增了词库, 如何让 jcseg 重新加载更改的词库呢?

1. 让 jcseg 自动轮询检测词库更新并且加载:
(1). 配置 jcseg.properties 中的 lexicon.autoload=1
(2). 配置 jcseg.properties 中的 lexicon.polltime=60 #为你预订的轮询时间(默认 60 秒)
(3). 告诉 jcseg 你需要重新加载的词库:
词库目录下有个 lex-autoload.todo 文件, 使用文本编辑器打开, 将你需要重新加载的词库名称一行一个写入即可, jcseg 会自动按照 lex-autoload.todo 最后修改时间来判断是否重新载入给定的词库文件. 完成 lex-autoload.todo 里面的词库加载后, jcseg 会自动清空里面的内容, 你也可以通过查看里面的内容来断定 jcseg 是否完成了给定词库的加载工作.

例如: 我给 lex-main.lex 和 lex-unitx.lex 加入了新词条.
使用文本编辑器打开 lex-autoload.todo 文件, 把 lex-main.lex 做为该文件的第一行, 把 lex-units.lex 作为第二行, 保存即可, 一段时间后(由 jcseg.properties 中的 lexicon.polltime 决定)指定的词库会被重新加载, 加载完后该文件会被清空.

2. 利用 jcseg 词库 API:
注: 这种情况下你需要确保 jcseg 的词库对象为同步的. 即, 通过:

```
ADictionary dic = DictionaryFactory
                .createDefaultDictionary(config, true);
```

来创建同步词库.

相信 ADictioanry.`loadFromLexiconDirectory` 和 `ADictionary.loadFromLexiconFile` 已经给定了你足够的信息来加载词库.
开发者只需要写一个程序适当的调用这个两个 API 即可, 至于通过何种方式来调用你的程序, 那取决于你的系统. 例如: 你可以做成一个网页来调用你的程序, 适当的时候访问指定网页来完成该工作.

九. 词库管理工具:

从 jcseg 1.9.1 开始, 里面自带了两个词库管理工具:
1. 简繁体相互转换:

```
Usage: java -jar jcseg-dicst.jar {tpy} {src} {dst}
```

{tpy}: 表示类型, 0 – 简体转换为繁体, 1 – 繁体转简体
{src}: 来源词库目录, 如果{tpy}为 0 则表示简体词库路径, 反之表示繁体词库路径.
{dst}: 结果存放目录, 表示将转换后的结果存放到指定目录.
例如: 将/home/chenxin/sim 下的简体 jcseg 词库(格式: lex-xxxx.lex)转换为繁体词库存放到/home/chenxin/tra 下:

```
java -jar jcseg-dicst.jar 0 /home/chenxin/sim /home/chenxin/tra
```

2. 词库合并工具:

```
Usage: java -jar jcseg-dicmerge.jar {dstdir} {srcdir1} {srcdir2} ...
```

{dstdir}: 合并之后的词库存放目录
{srcdir1}: 来源词库 1 目录
{srcdir2}: 来源词库 2 目录
...
{srcdirn}: 来源词库 n 目录
例如: 将 1 中得到的简体词库和繁体词库合并成一份词库放到/home/chenxin/mix 中:

```
java -jar jcseg-dicmerge.jar /home/chenxin/mix /home/chenxin/sim
/home/chenxin/tra
```

“词库合并”会将全部来源词库中的相同词库合并, 并且将相同词库中的词条合并, 同词条的词性和同义词合并. 来源词库可以只有一个, 那么就可以将词库去重, 并且按照自然顺序排序.
Jcseg-1.9.1 中的简繁体混合词库, 就是用”转换工具”从简体转换得到一份繁体词库, 然后再将简体和繁体合并得到的.

十. 更多文档:

1. Wiki 文档:
https://code.google.com/p/jcseg/w/list
2.jcseg.properties 文档说明:
https://code.google.com/p/jcseg/wiki/JcsegPropertiesFile
3.jcseg 词库类别以及含义:
https://code.google.com/p/jcseg/wiki/JcsegLexiconType

十一. 联系作者:

1. 作者信息: 陈鑫 – 网名: 狮子的魂
2. 电子邮件: chenxin619315@gmail.com

十二. 更多开源项目:

1. C 语言实现的高性能开源中文分词器 – friso
   http://code.google.com/p/friso
2. 基于 friso 实现的开源 php 中文分词扩展 – robbe
   http://code.google.com/p/robbe
3. 开源跨平台多媒体教学软件 – jteach
   http://code.google.com/p/jteach