Friso 开发帮助文档

(注:该文档只适合 friso 1.2 及以上的版本)

一. 关于 *friso:*

Friso 是使用 c 语言开发的一款中文分词器，使用流行的 mmseg 算法实现。完全基于模块化设计和实现，可以很方便的植入到其他程序中，例如：MySQL，PHP 等。源码无需修改就能在各种平台下编译使用，加载完 20 万的词条，内存占用稳定为 14.5M。

官方首页: https://code.google.com/p/friso

下载地址:

老版本：https://code.google.com/p/friso/downloads/list

新版本：http://sourceforge.net/projects/friso/files/?source=navbar

Friso 最新版本功能说明：（可以略过）

1。目前最高版本：friso 1.6.1，同时支持对 **UTF-8/GBK** 编码的切分。

2。三种切分模式：

(1). 简易模式：FMM 算法，适合速度要求场合。

(2). 复杂模式- MMSEG 四种过滤算法，具有较高的岐义去除，分词准确率达到了 98.41%。

(3). (!New)检测模式：只返回词库中已有的词条，很适合某些应用场合。(1.6.1 版本开始)

请参考本算法的原作：http://technology.chtsai.org/mmseg/。

3。支持自定义词库。在 dict 文件夹下，可以随便添加/删除/更改词库和词库词条，并且对词库进行了分类。

4。简体/繁体/简体混合支持, 可以方便的针对简体，繁体或者简繁体切分。同时还可以以此实现简繁体的相互检索。

5。支持中英/英中混合词的识别(维护词库可以识别任何一种组合)。例如：卡拉 ok, 漂亮 mm, c 语言，IC 卡，哆啦 a 梦。

7。很好的英文支持，英文标点组合词识别, 例如 c++, c#, 电子邮件，网址，小数，百分数。

8。(**!New**)自定义保留标点：你可以自定义保留在切分结果中的标点，这样可以识别出一些复杂的组合，例如：c++, k&r，code.google.com。

9。(**!New**)复杂英文切分的二次切分：默认 Friso 会保留数字和字母的原组合，开启此功能，可以进行二次切分提高检索的命中率。例如：qq2013 会被切分成：qq/ 2013/ qq2013。

10。支持阿拉伯数字/小数基本单字单位的识别，例如 2012 年，1.75 米，5 吨，120 斤，38.6℃。

11。自动英文圆角/半角，大写/小写转换。

12。同义词匹配：自动中文/英文同义词追加. (需要在 friso.ini 中开启 friso.add_syn 选项)。

13。自动中英文停止词过滤。(需要在 friso.ini 中开启 friso.clr_stw 选项)。

14。多配置支持, 安全的应用于多进程/多线程环境。

15。提供 friso.ini 配置文件, 可以依据你的需求轻松打造适合于你的应用的分词。

每个版本的升级的详细功能变化，请查看附件里面的 **CHANGES.md** 文件。

二. 安装 *friso:*

先到 friso 官方网站下载最新版本的 friso: friso-{version}-src-dict.zip, {version} 表示版本号, 下同.

解压 friso-{version}-src-dict.zip 到{path}, 设{path}为你的解压后 friso 的根目录, 下同:

1. Linux, 在终端 **cd** 到 **{path}/src** 目录后，然后依次运行：

make

sudo make install

注：如果是 **64** 位的系统，请将 **/usr/lib/libfriso.so** 拷贝一份到 **/usr/lib64** 中

2. WinNT:

(1). 使用 VS 编译得到 dll 和 lib 文件，具体可以参考 Friso 讨论：
http://www.oschina.net/question/853816_135216

(2). 使用 cygwin 从源码编译, 安装好 cygwin 后, 删除原有的 Makefile, 更改 Makefile.cygwin 为 Makefile, 打开 cygwin 的终端, cd 到{path}/src 下:

运行: make

在{path}/src 下即可得到 friso.exe 和 friso.dll

三. 配置 *friso:*

Friso 要做的配置工作很简单: 打开{path}目录, 找到 friso.ini 配置文件, 使用文本编辑器打开即可.

找到 friso.lex_dir, 修改其值为词库目录绝对地址, 并且必须以”/”结尾. 即:

friso.lex_dir = {path}/dict/GBK 或者 **UTF-8/**

例如: (回想第二步)假设你的 friso 解压在/opt/friso 下，使用 UTF-8 编码, 则:

friso.lex_dir = /opt/friso/dict/UTF-8/

friso.ini 配置文件**:(**可以不用理会**)**

```
#Friso 使用的切分编码。(0 表示 UTF-8, 1 表示 GBK, Friso-1.6.0)
friso.charset = 0

#词库绝对路径(注意词库分为 UTF-8 和 GBK)
friso.lex_dir = /c/products/friso/dict/UTF-8/

#最大匹配长度 ( 切分得到的词条的最大长度，建议：5-7 之间 )
friso.max_len = 5

#是否开启中文姓名识别(目前还不支持)
friso.r_name = 1

#最大英中混合词中文词数 ( 例如：b 超，中中文词数为 1 )
friso.mix_len = 2

#中文姓氏修饰词长度 ( 暂时无用 )
friso.lna_len = 1

#是否追加同义词 ( 1 表示开启，0 关闭 )
friso.add_syn = 1

#是否保留不识别的词条 (1 保留, 0 直接过滤)
#@date 2013-06-13
friso.keep_urec = 0

#是否启用 sphinx 定制输出(1 开启, 0 关闭)
#@date 2013-10-25
friso.spx_out = 0

#是否过滤停止词 ( 1 开启，0 关闭 )
friso.clr_stw = 0

#开启复杂英文的二次切分 (检索领域，建议开启，Friso-1.6.0)
#开启后：qq2013 会被切分成：qq/ 2013/ qq2013
friso.en_sseg = 1
```

```
#英文二次切分 sub Token 最小长度 (Friso-1.6.0)
#英文二次切分中，最小的子词条长度。friso 会检测每个英文 token，如果发现有多类字符，例如：数字，标点，字母，就会自动二次切分。例如：A2014，会被切分成：a/2014/ a2014，该选项就是用来限制最小的子切分长度，因为默认为 2，所以 A 过滤掉，也就是最终切分结果为：2014/ a2014
friso.st_minl = 2

#英文切分过程中默认保留的标点 (Friso-1.6.0)
#除非在词条的开始和结尾(%除外)，其余在任何位置都会保留，利于识别：k&r, c++, c#这类词条。
friso.kpuncs = @%.#&+

#用于姓名识别中的阈值. ( 暂时无用 )
friso.nthreshold = 2000000

#切分模式(1-简易模式, 2-复杂模式, 3 检测模式)
friso.mode = 2
```

四. 运行测试程序:

1. Linux, 在终端直接运行:

friso -init {path}/friso.ini

2. WinNT: (注意编码要设置为 **GBK**)

(1). cygwin 编译的, 在 cygwin 终端直接运行:

./friso -init {path}/friso.ini

(2). 没有 cygwin, 拷贝{path}/lib/win32 下的 friso.dll 到环境路径 path 下后, 载入 friso.dll, 然后使用 vc 或者 vs 编译运行{path}/src/tst-friso.c.

运行成功后你会看到如下的操作界面:

```
friso initialized in 0.160000sec
+------------------------------------------------------------------------------+
| friso - a chinese word segmentation writen by c.             |
| bug report email - chenxin619315@gmail.com.                  |
```

```
| or: visit http://code.google.com/p/friso.                              |
|    java edition for http://code.google.com/p/jcseg                     |
| type 'quit' to exit the program.                                       |
+------------------------------------------------------------------------+
friso>>
```

在提示 friso>>后输入你要分词的内容按 Enter 即可.

例如:

```
friso>> 研究生命起源, i love c++.
分词结果:
研究/ 琢磨/ 研讨/ 钻研/ 生命/ 起源/ ,/ i/ love/ c++/ ./
Done, cost < 0.000000sec
```

注意 1：提示界面的第一行有个: friso initialized in 0.160000sec, 如果是 friso initialized in 0.000000sec, 那就一定没有配置好, 也就是 friso 没有正确的加载词库, 确保 friso.ini 中的 friso.lex_dir 指向正确的词库目录。

注意 2: 如果直接使用 WinNT 下的 cmd 运行测试程序，注意要在 friso.ini 中设置编码为 GBK，也就是 friso.charset = 1, 同时设置词库路径为 GBK 词库路径。

五. *PHP* 扩展:

先按照上面的步骤安装 friso

1. 编译安装 friso php 扩展:

(1). cd {PATH}/binding/php

(2). 运行 phpize （需要安装 php-dev）

(3). 运行 ./configure

(4). 更改生成的 Makefile: 打开生成的 Makefile，在 friso.lo 和 friso.la 目标内容的-o 选项前加入 “-lfriso”来添加对 libfriso.so 的引用。

(5). 运行 make

(6). 运行 sudo make install

2. 配置 php.ini:

找到你的 php.ini 配置文件（通常在/etc/php5/下），打开。

在 extension 区域加入如下内容:

```
extension=friso.so  (friso.dll for winnt)
```

在 setting 前面加入:

```
[Friso]
; Defaul setting for friso
```

friso.ini_file=friso.ini 配置文件绝对地址
friso.ini_file 指向正确的 friso.ini 配置文件的地址。

3. 重启服务器：
重启你的 web 服务器，apache 或者 nginx or whatever

4. 检测配置是否生效：
建立一个 php 文件，里面调用 phpinfo()函数。
如果配置成功，运行该页面你将找到如下部分：

friso

Friso Support	enabled
Version	1.6.1
Bug Report	chenxin619315@gmail.com
Home page	http://code.google.com/p/friso

Directive	Local Value	Master Value
friso.ini_file	/c/products/friso/friso.ini	/c/products/friso/friso.ini

5. 运行 demo 程序：
{PATH}/binding/php/demo 下有三个 php 文件：
(1). friso.fun.php　　friso 绝大部分函数的使用 demo
(2). gbk.demo.php　　gbk 编码的分词测试程序
(3). utf8.demo.php　　UTF8 编码的分词测试程序
对应的.demo.php 需要在 friso.ini 中设置 friso.charset 并且加载对应编码的词库。

6. 常量说明：
friso 内部定义了很多常量，具体和含义如下：
FRISO_SIMPLE　　friso 简易切分模式 用在 friso_split 函数的 mode 配置中
FRISO_COMPLEX　　friso 复杂切分模式 ---
FRISO_DETECT　　friso 检测切分模式 ---

FRISO_RET_WORD　切分结果中返回词条内容
FRISO_RET_TYPE　　切分结果中返回词条类别
FRISO_RET_OFF　　切分结果中返回词条的偏移量
FRISO_RET_LEN　　切分结果中返回词条的优化后的长度

FRISO_RET_RLEN　切分结果中返回词条的优化前的长度
FRISO_RET_POS　切分结果中返回词条的词性

FRISO_TYP_CJK　返回的词条为 CJK 字符
FRISO_TYP_ECM　返回的词条为英中混合词
FRISO_TYP_CEM　返回的词条为中英混合词
FRISO_TYP_EPUN　返回的词条为英文标点混合词
FRISO_TYP_PUN　返回的词条为标点符号
FRISO_TYP_UNK　返回的词条为不识别词条
FRISO_TYP_OTR　返回的词条为其他类别词条

7. 主要函数说明：

friso.so 中包含了如下函数：

(1). string friso_charset()

返回 friso 当前设置的编码：

返回值：UTF8 或者 GBK

(2). string friso_version()

返回 friso 当前的版本号，例如：1.6.1

(3). string friso_ucode_utf8(long)

将给定的 unicode 编码数字转换为字符串(当个字)。例如：20013->中

(4). long friso_utf8_ucode(string)

将给定的当个字转换为 unicode 编码序号。例如：中->20013

(5). Array friso_split(string, Array, long);

这个是 friso 的核心函数也是最复杂的函数。

string: 要被切分的字符串

array: 自定义配置，可以为 NULL，表示使用 friso.ini 中的全局配置。

Long: 切分结果返回选项，指定多个使用“位或”联合。

返回结果为切分好的词条数组。

(1). 其中自定义配置如下：（具体含义请参考上面的 friso 的配置）

可以使用任意数量的配置选项，key 为 friso.ini 中的配置(friso.charset 除外)，值和 friso.ini 中提的供可选值一样。

例如：复杂模式，最大匹配长度为 5, 去除停止词，追加同义词配置如下：

```
array(
    'max_len'=>5,              //最大匹配长度
    'add_syn'=>1,
    'clr_stw'=>1,
    'mode'=>FRISO_COMPLEX
);
```

(2). 切分结果的返回：

你可以自定义 friso 返回切分词条以及相关信息，主要如下：

FRISO_RET_WORD　　-词条内容，例如：”研究“

FRISO_RET_TYPE　　-词条类别，对应上面包含 TYP 的常量。

FRISO_RET_LENGTH　-词条长度，friso 优化后的词条的字节数。

FRISO_RET_RLEN　　-词条真实长度，friso 优化前的实际字节数（全角转半角）

FRISO_RET_OFF　　-词条在原文中的偏移量，词条在原文的开始位置

FRISO_RET_POS　　-词条词性说明符号，例如：n(待实现)

例如：返回词条，长度和偏移量，则设置第三个参数为：

FRISO_RET_WORD | FRISO_LENGTH | FRISO_OFF

详细的使用例子，请查看 friso.fun.php

六. *Sphinx* 集成:

插件开发中。。。

七. 二次开发:

要使用 friso 来进行分词, 你需要两个对象: **friso_t**(friso_entry)对象和一个 **friso_task_t**(friso_task_entry)对象. 两者都在 friso.h 头文件中定义的:

(可以先看下面的”一个完整的例子”)

1. friso_t/friso_config_t 对象：(1.6.0 开始)

定义：

```
/* friso entry.*/
typedef struct {
```

```
        friso_dic_t dic;                //friso dictionary
        friso_charset_t charset;        //friso charset.
    } friso_entry;
    typedef friso_entry * friso_t;
    //其中的重点就是 dic, 也就是 friso 的词库.

    /*friso_config_entry 配置实例*/
    struct friso_config_struct {
        ushort_t max_len;     //the max match length (4 - 7).
        ushort_t r_name;      //1 for open chinese name recognition 0 for close
it.
        ushort_t mix_len;     //the max length for the CJK words in a mix string.
        ushort_t lna_len;     //the max length for the chinese last name adron.
        ushort_t add_syn;     //append synonyms tokenizer words.
        ushort_t clr_stw;     //clear the stopwords.
        ushort_t keep_urec;  //keep the unrecongnized words.
        ushort_t spx_out;     //use sphinx output customize.
        ushort_t en_sseg;     //start the secondary segmentation.
        ushort_t st_minl;     //min length of the secondary segmentation token.
        uint_t nthreshold;    //the threshold value for a char to make up a
chinese name.
        friso_mode_t mode;  //Complex mode or simple mode



        //pointer to the function to get the next token
        friso_token_t (*next_token) (friso_t, struct friso_config_struct *,
friso_task_t);
        //pointer to the function to get the next cjk lex_entry_t
        lex_entry_t   (*next_cjk  ) (friso_t, struct friso_config_struct *,
friso_task_t);
        char kpuncs[_FRISO_KEEP_PUNC_LEN]; //keep punctuations buffer.
};
typedef struct friso_config_struct friso_config_entry;
typedef friso_config_entry * friso_config_t;
```

创建和初始化：

friso 内部提供了 api 来创建并且初始化 friso_entry 的函数:

(1). 单独创建并且设置:

```
//创建 friso_t
friso_t friso = friso_new();

//创建词库 dic(并没有加载词库)
friso_config_t config = friso_new_config();

//从指定的 friso.ini 配置文件中初始化 friso 和 config
//__path__为 friso.ini 配置文件的地址
//成功返回 1, 失败返回 0
friso_init_from_ifile(friso, config, __path__);

//自定义切分模式(简易，复杂，检测模式)
//__FRISO_SIMPLE_MODE__
//__FRISO_COMPLEX_MODE__
//__FRISO_DETECT_MODE__
friso_set_mode(config, __FRISO_COMPLEX_MODE__);
```

释放:

friso_t 实例用完后需要使用如下 api 来释放:

```
//释放 friso_t 实例
friso_free(friso);
friso_free_config(config);
```

2. friso_task_t 对象:

定义:

```
typedef struct {
    fstring text;           //text to tokenize
    uint_t idx;             //start offset index.
```

```
    uint_t length;          //length of the text.
    uint_t bytes;           //latest word bytes in C.
    uint_t unicode;         //latest word unicode number.
    friso_link_t pool;      //task pool.
    string_buffer_t sbuf; //string buffer. (Friso-1.6.0)
    friso_hits_t hits;      //token result hits.
    char buffer[7];         //word buffer. (1-6 bytes for an utf-8 word in C).
} friso_task_entry;
typedef friso_task_entry * friso_task_t;
```

text 指向需要被切分的 utf-8 编码的字符串.
idx 表示下一个切分的开始索引.
length 表示字符串的长度(字节).
hits 表示一个切分结果.
pool 切分结果缓冲池(一个链表).
其他的是一些为方便中间过程切分的辅助变量.

再来看下 hits(friso_hits_t)的结构:

```
typedef struct {
    uchar_t type;       //type of the word. (item of friso_lex_t) (Friso-1.6.0)
    uchar_t length;     //length of the token. (Friso-1.6.0)
    uchar_t rlen;       //the real length of the token.(Friso-1.6.0)
    char pos;           //part of speech.  (Friso-1.6.0)
    int offset;
    char word[__HITS_WORD_LENGTH__];
} friso_hits_entry;
typedef friso_hits_entry * friso_hits_t;
```

friso_hits_t 是用来保存一个切分结果.
type 是词条类别。
length 是词条长度。（Friso 内部优化之后）
rlen 词条真实长度。(Friso 内部优化前)
pos 词条词性。(待实现。)
offset 是这个切分到的词在整个字符串中的偏移量。
word 即为这个词。

创建：

同样的, friso 内部提供了 api 来创建 friso_task_t.

```
//创建一个分词任务实例
friso_task_t task = friso_new_task();

//给分词任务设置分词的内容.
fstring text = “研究生命起源”;
friso_set_text( task, text );
```

释放：

同样的, 用完的 friso_task_t 需要调用下面的 api 来释放:

```
//释放 friso_task_t 实例
friso_free_task( task );
```

3. 看一个完整的例子：

接下我们使用 friso_t 和 friso_task_t 来写一个完整的列子

详细查看源码中 **tst-friso.c** 完整的样板：

(注：从 1.6.1 开始，friso_hits_t 改名为了 friso_token_t)

```
//1. 创建和初始化资源
friso_t friso = friso_new();
friso_config_t config = friso_new_config();
//从指定的 friso.ini 文件中初始化 friso 和 config.
friso_init_from_ifile(friso, config, _ifile);

//2.创建分词任务&&设置分词内容
friso_task_t task = friso_new_task();
fstring text = “这里是要被分词的字符串”;
friso_set_text( task, text );
```

```
//3 .获取切分结果
----------------------------------------1.6.1 以前的版本
//friso_next 获取下一个切分结果
//得到的切分结果存放在 task->hits 中.
//通过 task->hits->word 的到切分的词条.
//通过 task->hits->offset 得到对应词条在原文中的偏移位置.
while ( ( friso_next( friso, config, task ) ) != NULL ) {
    //查看 friso_hits_t 可以获取更多信息。
    //printf("%s[%d]/ ", task->hits->word, task->hits->offset );
    printf("%s/ ", task->hits->word );
}

----------------------------------------------1.6.1 及以后的版本
//从 1.6.1 开始，为了方便或者，friso_hits_t 更改为 friso_token_t
不在使用 friso_next 来获取下一个切分结果，而是调用 config->next_token 来获取下一个切分结果，这样可以统一不同切分模式的入口。
while ( ( config->next_token( friso, config, task ) ) != NULL ) {
    //查看 friso_token_t 可以获取更多信息。
    //printf("%s[%d]/ ", task->token->word, task->token->offset );
    printf("%s/ ", task->token->word );
}

//4. 释放资源...
friso_free_task( task );
friso_free_config(config);
friso_free(friso);
```

注意: 在单线程环境下可以反复的利用创建的 friso_t, friso_config_t(1.5.0 以及以上)和 friso_task_t. 切分不同的内容的时候调用 friso_set_text(friso_task_t, fstring)来重置 friso_task_t 的切分内容即可. {path}/src/tst-friso.c 是一个完整的例子.

而, 在多线程环境下: 不同线程共用一个 friso_t, 每个线程都创建一个 friso_task_t. 具体例子, 可以查看 binding/php/下的 friso 的 php 扩展.

八. 词库管理:

Friso 内部对词库进行了分类, 在管理词库前你需要先了解这些分类:

friso 词库类别：

```
typedef enum {
    __LEX_CJK_WORDS__ = 0,          //普通 CJK 词库
    __LEX_CJK_UNITS__ = 1,          //CJK 单位词库
    __LEX_ECM_WORDS__ = 2,          //英中混合词(例如: b 超)
    __LEX_CEM_WORDS__ = 3,          //中英混合词(例如: 卡拉 ok).
    __LEX_CN_LNAME__ = 4,           //中文姓氏
    __LEX_CN_SNAME__ = 5,           //中文单姓名词库
    __LEX_CN_DNAME1__ = 6,          //中文双姓名首字词库
    __LEX_CN_DNAME2__ = 7,          //中文双姓名尾字词库
    __LEX_CN_LNA__ = 8,             //中文姓氏修饰词词库
    __LEX_STOPWORDS__ = 9,          //停止词词库
    __LEX_ENPUN_WORDS__ = 10,       //英文和标点混合词库(例如: c++)
    __LEX_OTHER_WORDS__ = 15,       //无用
    __LEX_NCSYN_WORDS__ = 16        //无用
} friso_lex_t;
```

再来看看 friso.lex.ini 配置文件:

```
#main lexion
__LEX_CJK_WORDS__ :[
    lex-main.lex;
    lex-admin.lex;
    lex-chars.lex;
    lex-cn-mz.lex;
    lex-cn-place.lex;
    lex-company.lex;
    lex-festival.lex;
    lex-flname.lex;
    lex-food.lex;
    lex-lang.lex;
    lex-nation.lex;
    lex-net.lex;
    lex-org.lex;
```

```
    #add more here
]
#single chinese unit lexicon
__LEX_CJK_UNITS__ :[
    lex-units.lex;
]
#chinese and english mixed word lexicon like "b 超".
__LEX_ECM_WORDS__:[
    lex-ecmix.lex;
]
#english and chinese mixed word lexicon like "卡拉 ok".
__LEX_CEM_WORDS__:[
    lex-cemix.lex;
]
#chinese last name lexicon.
__LEX_CN_LNAME__:[
    lex-lname.lex;
]
#single name words lexicon.
__LEX_CN_SNAME__:[
    lex-sname.lex;
]
#first word of a double chinese name.
__LEX_CN_DNAME1__:[
    lex-dname-1.lex;
]
#second word of a double chinese name.
__LEX_CN_DNAME2__:[
    lex-dname-2.lex;
]
#chinese last name decorate word.
__LEX_CN_LNA__:[
    lex-lna.lex;
]
#stopwords lexicon
__LEX_STOPWORDS__:[
```

```
        lex-stopwords.lex;
    ]
    #english and punctuation mixed words lexicon.
    __LEX_ENPUN_WORDS__:[
        lex-en-pun.lex;
    ]
```

格式如下:

```
词库类别关键字: [
    词库文件;
]
```

上面的 10 个词库类被关键字分别对应于 friso 的 10 个词库类别, []中的内容就是该类别的词库文件, 一个类别可以有多个词库文件. 类别是系统定义的, 不能随便添加.

1. 加入新词库文件**:**

首先确认你要加入的词库文件的类别.

例如: 我想添加一个词库文件专门用来存储植物的名字, 在{path}/dict/下新建 lex-pname.lex, 然后按照一个词条一行的规则加入词条到 lex-pname.lex 来完善该词库.

接下来你还有一个重要的步骤就是将该词库归类到 friso.lex.ini 中去, 通常的词库都是 CJK 词库, 也就是将 lex-pname.lex 作为一行加入到:

```
    __LEX_CJK_WORDS__:[
        lex-main.lex;
        lex-admin.lex;
        lex-chars.lex;
        lex-cn-mz.lex;
        lex-cn-place.lex;
        lex-company.lex;
        lex-festival.lex;
        lex-flname.lex;
        lex-food.lex;
        lex-lang.lex;
        lex-nation.lex;
        lex-net.lex;
        lex-org.lex;
        lex-pname.lex;
#add more here
```

```
]
```

新词库文件的加入工作就 bingo 了.

2. 在给定词库文件中加入新词条**:**

这个工作做起来太简单了, 找到对应的词库文件, 使用文本编辑器打开, 将要加入的词条按照下面的格式作为一行加入即可. (Tip: 加入前先确认下相同的词条不存在, 重复存在也没关系, 只不过会浪费磁盘空间并且会影响词库的加载时间).

Friso 词库词条格式:

```
词条/同义词集合
```

同义词没有使用 null 代替, 多个同义词使用英文逗号隔开.

例如: 研究

研究/琢磨,研讨,钻研

3. 繁体/简繁体混合支持: (friso-1.5.0 以及以上版本):

在 friso 官网下载最新的全部词库, simplified 是简体词库, traditional 是繁体词库, mixed 是简繁体混合词库, 依据你的需求选择对应的词库就可以了.

九. 联系作者**:**

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

Blog: http://www.lionsoul.org

十. 更多开源软件**:**

1. java 开源中文分词分词器 – jcseg
   http://code.google.com/p/jcseg
3. 开源跨平台多媒体教学软件 – jteach
   http://code.google.com/p/jteach