第一章 德塔语言图灵工程.

德塔语言图灵工程 API 说明书 V_10_6_1

作者: 罗瑶光
ID:430181198505250014
浏阳德塔软件开发有限公司
2019 年 4 月 2 日

第一节 研发说明

1. 起源动机

作者2018 年前一直用 lucene 包分词. 后来在处理古代中医学(德塔养疗经)书发现20 亿字无法迅速处理. 当时 lucene 每秒分中文 20 万字,核心都是闭源. 于是作者通过自己语文教育能力和对中文的理解编辑成函数.
2018 年9 月每秒分词2100 万. 于是开始优化.
2018 年10 月每秒分词1900 万, 添加了9 种词性根据象形语言文字文学分析.
2018 年11 月优化到每秒1700 万, 添加了病句和英文字符的提取做契字分析.
2018 年12 月开始国际sonar 的弱模式优化,用性能换认知, 每秒分词1300 万,这个世界记录已经持续4 个月. Deta 为了记录这个历史时刻,将该项目免费版本发布在 Github Apacher 2.0 license.
Deta 与华夏知识产权合作,于2019 年3 月30 日已经提交中华人民共和国专利申请.
VPCS Chinese Word Segmentation 23:29 2019-03-31 再次刷新世界记录.
每秒纯中文分词1400 万.(字典森林区域化优化)
VPCS Chinese Word Segmentation 01:31 2019-04-02 再次刷新世界记录.
每秒纯中文分词1500 万.(离散交集化 简优化)
(Chinese Word Segmentation 是纯中文分词的意思)

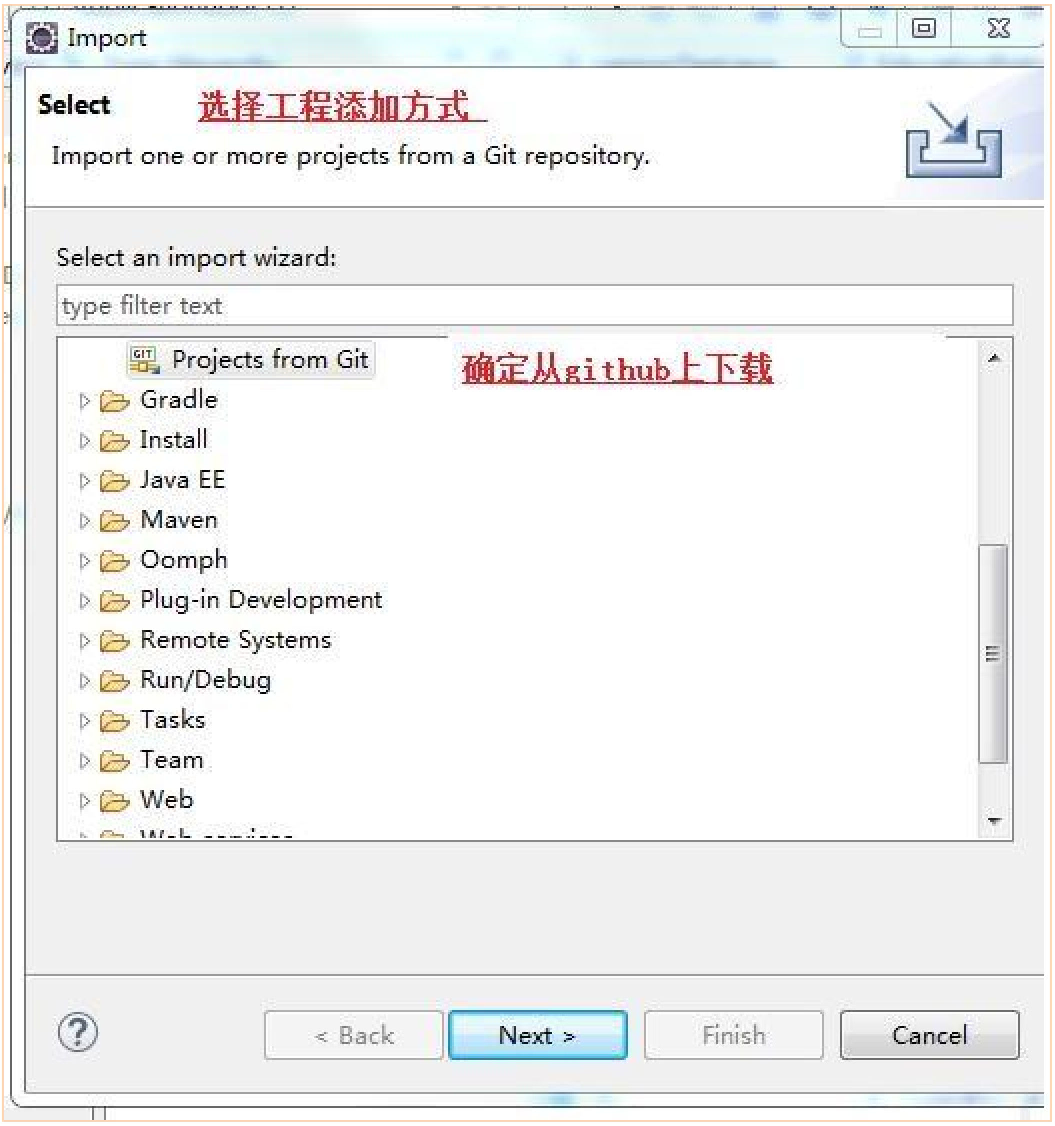
2. 简介

Deta 图灵工程作为 Deta 人工智能的核心组成部份主要任务就是极为快速词语处理. 主要用在文本的快速词语分离, 词性分析, 自然语言处理和心理学领域 .

3 使用方法

3.1 下载 java 开发软件:
Eclipse: https://www.eclipse.org/
Intellij: https://www.jetbrains.com/idea/

3.2 导入 deta 图灵 api (API 是类库,接口 的意思, select 是选择 的意思)


Import

Select 选择工程添加方式

Import one or more projects from a Git repository.

Select an import wizard:

type filter text

Projects from Git 确定从github上下载

Gradle

Install

Java EE

Maven

Oomph

Plug-in Development

Remote Systems

Run/Debug

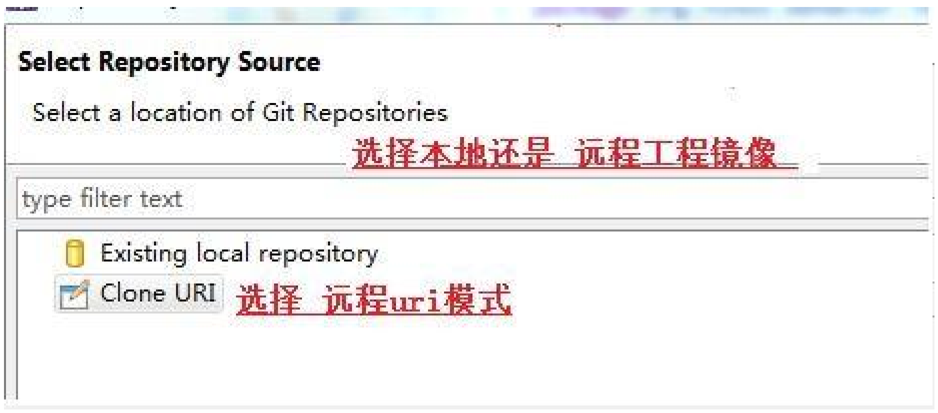
Tasks

Team

Web

< Back | Next > | Finish | Cancel

3.3 点URI (uri 是互联网传输的一种协议规范关键字)


Select Repository Source

Select a location of Git Repositories

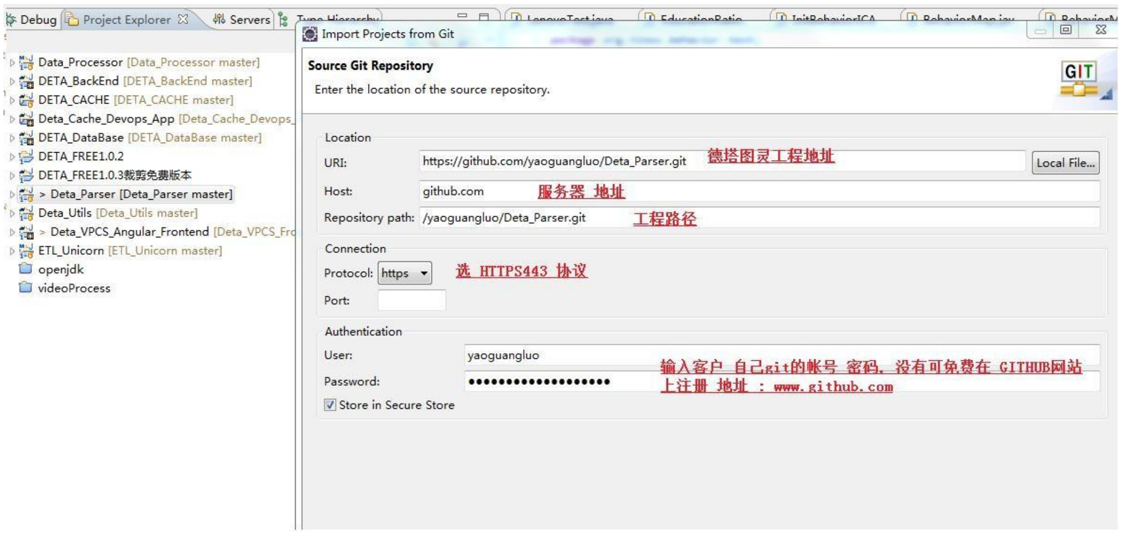
选择本地还是 远程工程镜像

type filter text

Existing local repository

Clone URI 选择 远程uri模式

3.4 输入 Git 导入目标地址 (git 是版本持续化控制软件, repository 是 git 工程的下载标识, host 是远程 主机, repository path 是 git 工程 在主机上下载链接, protocol 是是通信协议, port 是端口, authentication 是密钥, user 是帐户名, password 是密码, store in secure store 是记录保存)



Debug | Project Explorer | Servers | Type Hierarchy

- Data_Processor [Data_Processor master]
- DETA_BackEnd [DETA_BackEnd master]
- DETA_CACHE [DETA_CACHE master]
- Deta_Cache_Devops_App [Deta_Cache_Devops_
- DETA_DataBase [DETA_DataBase master]
- DETA_FREE1.0.2
- DETA_FREE1.0.3裁剪免费版本
- > Deta_Parser [Deta_Parser master]
- Deta_Utils [Deta_Utils master]
- > Deta_VPCS_Angular_Frontend [Deta_VPCS_Fro
- ETL_Unicorn [ETL_Unicorn master]
- openjdk
- videoProcess

Import Projects from Git

Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/yaoguangluo/Deta_Parser.git 德塔图灵工程地址 [Local File...]

Host: github.com 服务器 地址

Repository path: /yaoguangluo/Deta_Parser.git 工程路径

Connection

Protocol: https 选 HTTPS443 协议

Port:

Authentication

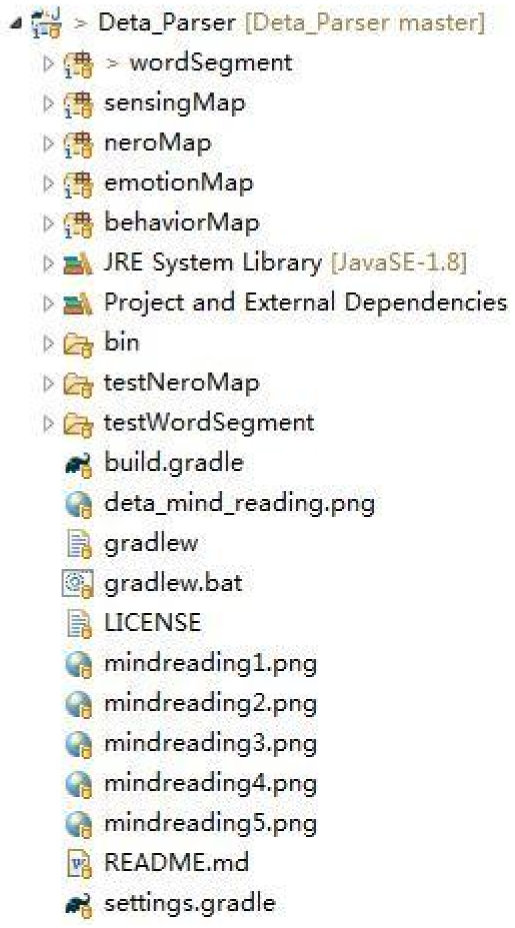
User: yaoguangluo

Password: ••••••••••••••••••••

输入客户 自己git的帐号 密码, 没有可免费在 GITHUB网站上注册 地址 : www.github.com

[✓] Store in Secure Store

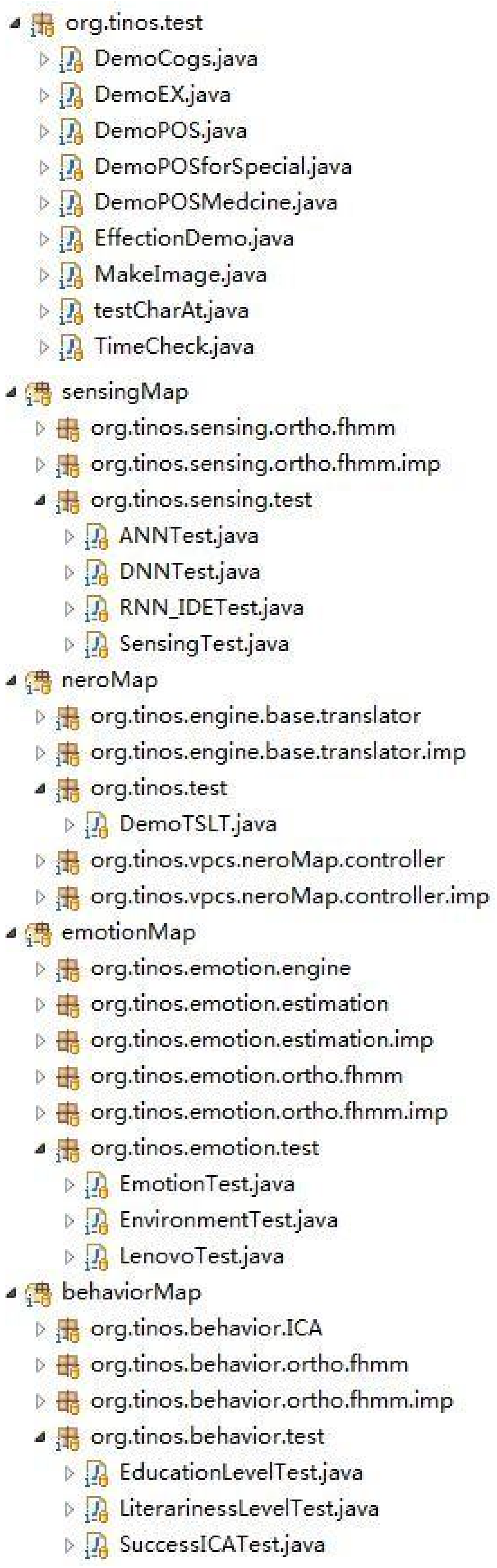
3.5 生成 eclipse 工程 因为是无插件底层源码，所以可以自由集成为pom, gradle, web,或者general 工程模式. (POM 是xml 形式的库标识 标识, gradle 是 模板形式, web 是web 2.0 动态java 工程, general 是



- > Deta_Parser [Deta_Parser master]
  - > wordSegment
  - sensingMap
  - neroMap
  - emotionMap
  - behaviorMap
  - JRE System Library [JavaSE-1.8]
  - Project and External Dependencies
  - bin
  - testNeroMap
  - testWordSegment
  - build.gradle
  - deta_mind_reading.png
  - gradlew
  - gradlew.bat
  - LICENSE
  - mindreading1.png
  - mindreading2.png
  - mindreading3.png
  - mindreading4.png
  - mindreading5.png
  - README.md
  - settings.gradle

普通java 工程)

3.6 运行例子就可以了 所有 demo 和 test 都是 可运行实例 (demo 是例子的意思, test 是测试的意思 鼠标右键,点运行就可以了.)

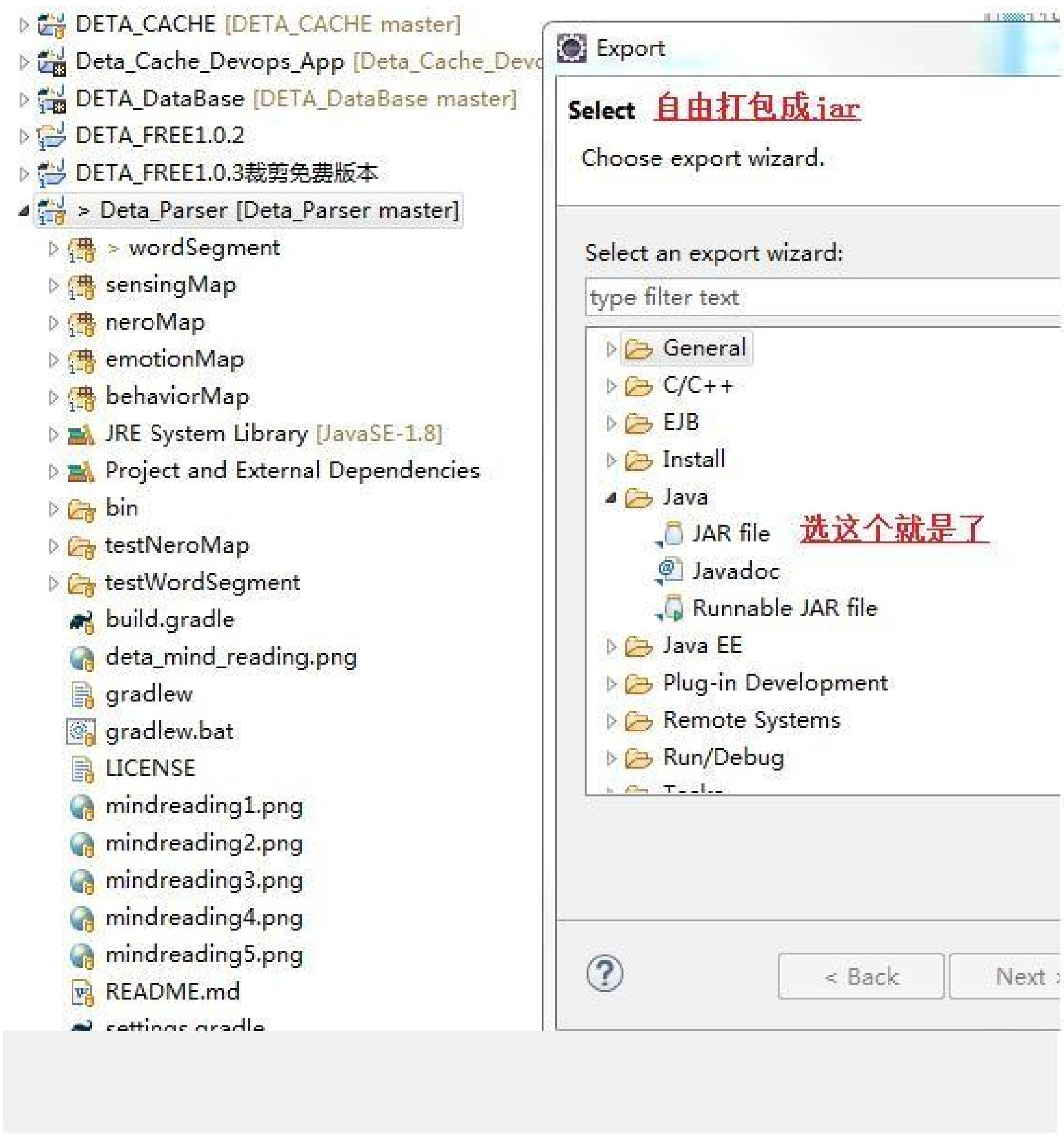


```
◢ org.tinos.test
    ▷ DemoCogs.java
    ▷ DemoEX.java
    ▷ DemoPOS.java
    ▷ DemoPOSforSpecial.java
    ▷ DemoPOSMedcine.java
    ▷ EffectionDemo.java
    ▷ MakeImage.java
    ▷ testCharAt.java
    ▷ TimeCheck.java
◢ sensingMap
    ▷ org.tinos.sensing.ortho.fhmm
    ▷ org.tinos.sensing.ortho.fhmm.imp
    ◢ org.tinos.sensing.test
        ▷ ANNTest.java
        ▷ DNNTest.java
        ▷ RNN_IDETest.java
        ▷ SensingTest.java
◢ neroMap
    ▷ org.tinos.engine.base.translator
    ▷ org.tinos.engine.base.translator.imp
    ◢ org.tinos.test
        ▷ DemoTSLT.java
    ▷ org.tinos.vpcs.neroMap.controller
    ▷ org.tinos.vpcs.neroMap.controller.imp
◢ emotionMap
    ▷ org.tinos.emotion.engine
    ▷ org.tinos.emotion.estimation
    ▷ org.tinos.emotion.estimation.imp
    ▷ org.tinos.emotion.ortho.fhmm
    ▷ org.tinos.emotion.ortho.fhmm.imp
    ◢ org.tinos.emotion.test
        ▷ EmotionTest.java
        ▷ EnvironmentTest.java
        ▷ LenovoTest.java
◢ behaviorMap
    ▷ org.tinos.behavior.ICA
    ▷ org.tinos.behavior.ortho.fhmm
    ▷ org.tinos.behavior.ortho.fhmm.imp
    ◢ org.tinos.behavior.test
        ▷ EducationLevelTest.java
        ▷ LiterarinessLevelTest.java
        ▷ SuccessICATest.java
```

3.7 网页 例子:

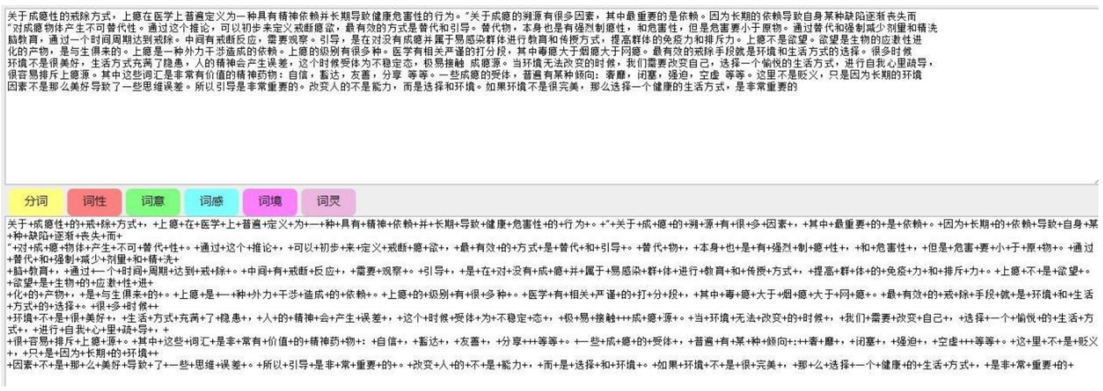
http://tinos.qicp.vip/data.html

3.8 可以任意 打包 jar 作为商业 库销售和集成.(jar 是java 的库的意思, 可运行,可扩展, 可集成, export 是打包输出的意思)



4 具体重要功能展示

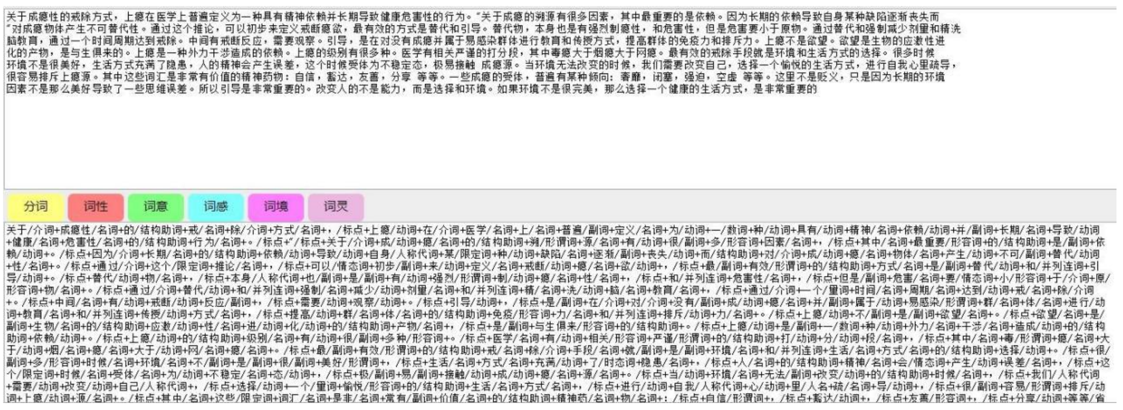
4.1 每秒 1500 万中文快速分词:



```
关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。“关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而
“对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗
脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进
化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候
环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触 成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，
很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境
因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的
```

```
关于+成瘾性+的+戒+除+方式+，+上瘾+在+医学+上+普遍+定义+为+一+种+具有+精神+依赖+并+长期+导致+健康+危害性+的+行为+。+“+关于+成+瘾+的+溯+源+有+很+多+因素+，+其中+最重要+的+是+依赖+。+因为+长期+的+依赖+导致+自身+某
+种+缺陷+逐渐+丧失+而+
“+对+成+瘾+物体+产生+不可+替代+性+。+通过+这个+推论+，+可以+初步+来+定义+戒断+瘾+欲+，+最+有效+的+方式+是+替代+和+引导+。+替代+物+，+本身+也+是+有+强烈+制+瘾+性+，+和+危害性+，+但是+危害+要+小+于+原+物+。+通过
+替代+和+强制+减少+剂量+和+精+洗+
+脑+教育+，+通过+一个+时间+周期+达到+戒+除+。+中间+有+戒断+反应+，+需要+观察+。+引导+，+是+在+对+没有+成+瘾+并+属于+易感染+群+体+进行+教育+和+传授+方式+，+提高+群+体+的+免疫+力+和+排斥+力+。+上瘾+不+是+欲望+。
+欲望+是+生物+的+应激+性+进+
+化+的+产物+，+是+与生俱来+的+。+上瘾+是+一+种+外力+干涉+造成+的+依赖+。+上瘾+的+级别+有+很+多种+。+医学+有+相关+严谨+的+打+分+段+，+其中+毒+瘾+大于+烟+瘾+大于+网+瘾+。+最+有效+的+戒+除+手段+就+是+环境+和+生活
+方式+的+选择+。+很+多+时候++
+环境+不+是+很+美好+，+生活+方式+充满+了+隐患+，+人+的+精神+会+产生+误差+，+这个+时候+受体+为+不稳定+态+，+极+易+接触+++成+瘾+源+。+当+环境+无法+改变+的+时候+，+我们+需要+改变+自己+，+选择+一个+愉悦+的+生活+方
式+，+进行+自我+心+里+疏+导+，+
+很+容易+排斥+上瘾+源+。+其中+这些+词汇+是非+常有+价值+的+精神药+物+：+自信+，+豁达+，+友善+，+分享+++等等+。+一些+成+瘾+的+受体+，+普遍+有+某+种+倾向+:++奢+靡+，+闭塞+，+强迫+，+空虚+++等等+。+这+里+不+是+贬义
+，+只+是+因为+长期+的+环境++
+因素+不+是+那+么+美好+导致+了+一些+思维+误差+。+所以+引导+是非+常+重要+的+。+改变+人+的+不+是+能力+，+而+是+选择+和+环境+。+如果+环境+不+是+很+完美+，+那+么+选择+一个+健康+的+生活+方式+，+是非+常+重要+的+
```

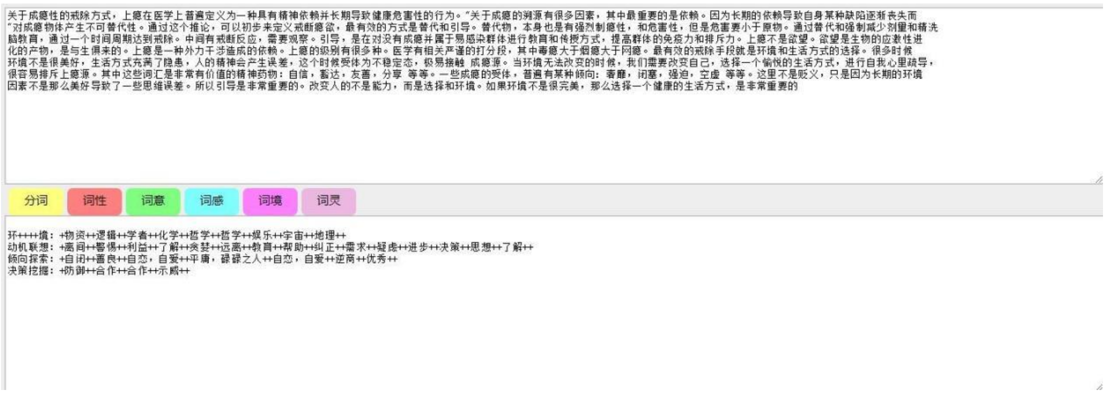
4.2 每秒 900 万中文词性标注



```
关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。"关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗
脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进
化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候
环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触 成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，
很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境
因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的
```

```
关于/介词+成瘾性/名词+的/结构助词+戒/名词+除/介词+方式/名词+，/标点+上瘾/动词+在/介词+医学/名词+上/名词+普遍/副词+定义/名词+为/动词+一/数词+种/动词+具有/动词+精神/名词+依赖/动词+并/副词+长期/名词+导致/动词
+健康/名词+危害性/名词+的/结构助词+行为/名词+。/标点+"/标点+关于/介词+成/动词+瘾/名词+的/结构助词+溯/形谓词+源/名词+有/动词+很/副词+多/形容词+因素/名词+，/标点+其中/名词+最重要/形容词+的/结构助词+是/副词+依
赖/动词+。/标点+因为/介词+长期/名词+的/结构助词+依赖/动词+导致/动词+自身/人称代词+某/限定词+种/动词+缺陷/名词+逐渐/副词+丧失/动词+而/结构助词+对/介词+成/动词+瘾/名词+物体/名词+产生/动词+不可/副词+替代/动词
+性/名词+。/标点+通过/介词+这个/限定词+推论/名词+，/标点+可以/情态词+初步/副词+来/动词+定义/名词+戒断/动词+瘾/名词+欲/动词+，/标点+最/副词+有效/形谓词+的/结构助词+方式/名词+是/副词+替代/动词+和/并列连词+引
导/动词+。/标点+替代/动词+物/名词+，/标点+本身/人称代词+也/副词+是/副词+有/动词+强烈/形谓词+制/动词+瘾/名词+性/名词+，/标点+和/并列连词+危害性/名词+，/标点+但是/副词+危害/名词+要/情态词+小/形容词+于/介词+原/
形容词+物/名词+。/标点+通过/介词+替代/动词+和/并列连词+强制/名词+减少/动词+剂量/名词+和/并列连词+精/名词+洗/动词+脑/名词+教育/名词+，/标点+通过/介词+一个/量词+时间/名词+周期/名词+达到/动词+戒/名词+除/介词
+。/标点+中间/名词+有/动词+戒断/动词+反应/副词+，/标点+需要/动词+观察/动词+。/标点+引导/动词+，/标点+是/副词+在/介词+对/介词+没有/副词+成/动词+瘾/名词+并/副词+属于/动词+易感染/形谓词+群/名词+体/名词+进行/动
词+教育/名词+和/并列连词+传授/动词+方式/名词+，/标点+提高/动词+群/名词+体/名词+的/结构助词+免疫/形容词+力/名词+和/并列连词+排斥/动词+力/名词+。/标点+上瘾/动词+不/副词+是/副词+欲望/名词+。/标点+欲望/名词+是/
副词+生物/名词+的/结构助词+应激/动词+性/名词+进/动词+化/动词+的/结构助词+产物/名词+，/标点+是/副词+与生俱来/形容词+的/结构助词+。/标点+上瘾/动词+是/副词+一/数词+种/动词+外力/名词+干涉/名词+造成/动词+的/结构
助词+依赖/动词+。/标点+上瘾/动词+的/结构助词+级别/名词+有/动词+很/副词+多种/形容词+。/标点+医学/名词+有/动词+相关/形容词+严谨/形谓词+的/结构助词+打/动词+分/动词+段/名词+，/标点+其中/名词+毒/形谓词+瘾/名词+大
于/动词+烟/名词+瘾/名词+大于/动词+网/名词+瘾/名词+。/标点+最/副词+有效/形谓词+的/结构助词+戒/名词+除/介词+手段/名词+就/副词+是/副词+环境/名词+和/并列连词+生活/名词+方式/名词+的/结构助词+选择/动词+。/标点+很/
副词+多/形容词+时候/名词+环境/名词+不/副词+是/副词+很/副词+美好/形谓词+，/标点+生活/名词+方式/名词+充满/动词+了/时态词+隐患/名词+，/标点+人/名词+的/结构助词+精神/名词+会/情态词+产生/动词+误差/名词+，/标点+这
个/限定词+时候/名词+受体/名词+为/动词+不稳定/名词+态/动词+，/标点+极/副词+易/副词+接触/动词+成/动词+瘾/名词+源/名词+。/标点+当/动词+环境/名词+无法/副词+改变/动词+的/结构助词+时候/名词+，/标点+我们/人称代词
+需要/动词+改变/动词+自己/人称代词+，/标点+选择/动词+一个/量词+愉悦/形容词+的/结构助词+生活/名词+方式/名词+，/标点+进行/动词+自我/人称代词+心/动词+里/人名+疏/名词+导/动词+，/标点+很/副词+容易/形谓词+排斥/动
词+上瘾/动词+源/名词+。/标点+其中/名词+这些/限定词+词汇/名词+是非/名词+常有/副词+价值/名词+的/结构助词+精神药/名词+物/名词+：/标点+自信/形谓词+，/标点+豁达/动词+，/标点+友善/形容词+，/标点+分享/动词+等等/省
```

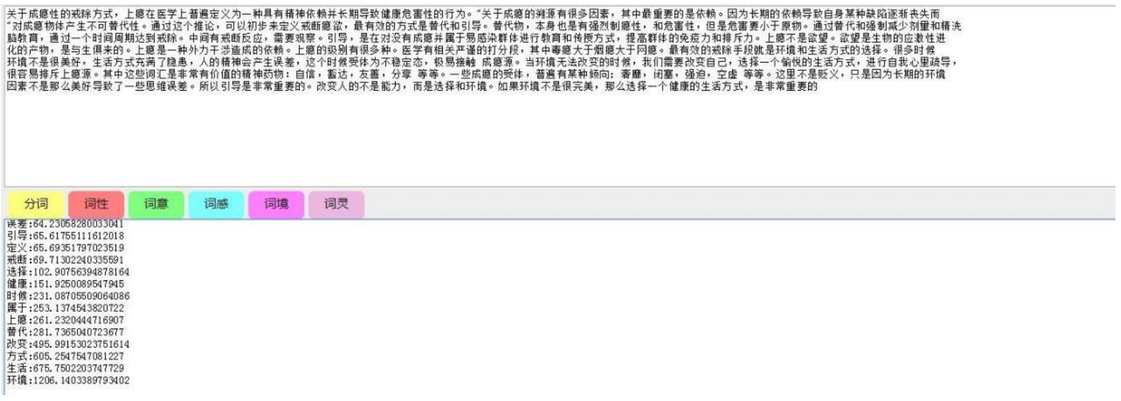
4.3 文字意义 分析



```
关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。"关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗
脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进
化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候
环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触 成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，
很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境
因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的
```

```
环++++境：+物资++逻辑++学者++化学++哲学++哲学++娱乐++宇宙++地理++
动机联想：+离间++警惕++利益++了解++贪婪++远离++教育++帮助++纠正++需求++疑虑++进步++决策++思想++了解++
倾向探索：+自闭++善良++自恋，自爱++平庸，碌碌之人++自恋，自爱++逆商++优秀++
决策挖掘：+防御++合作++合作++示威++
```

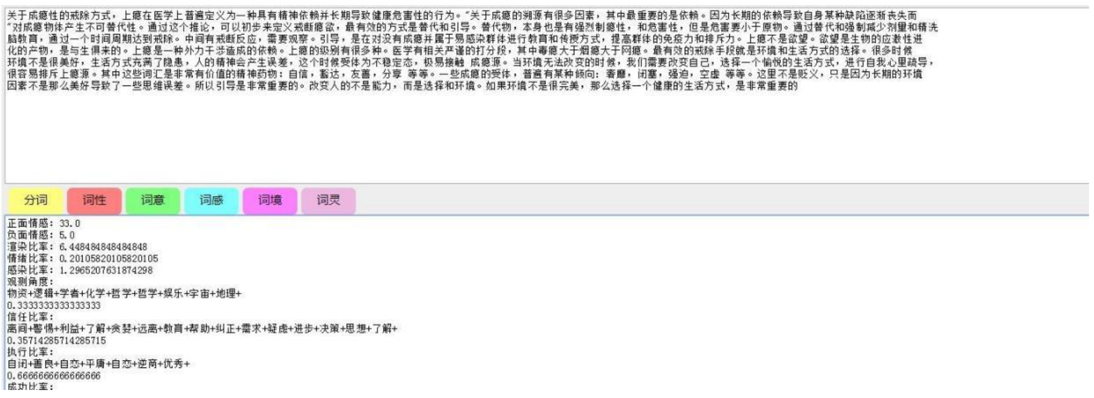
4.4 文字情感 分析



```
关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。"关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗
脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进
化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候
环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触 成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，
很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境
因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的
```

```
误差:64.2305828003304l
引导:65.61755111612018
定义:65.69351797023519
戒断:69.71302240335591
选择:102.90756394878164
健康:151.9250089547945
时候:231.08705509064086
属于:253.1374543820722
上瘾:261.2320444716907
替代:281.7365040723677
改变:495.99153023751614
方式:605.2547547081227
生活:675.7502203747729
环境:1206.1403389793402
```

4.5 文字语境 分析



```
关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。"关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗
脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进
化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候
环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触 成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，
很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境
因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的
```

```
正面情感: 33.0
负面情感: 5.0
渲染比率: 6.448484848484848
情绪比率: 0.20105820105820105
感染比率: 1.2965207631874298
观测角度:
物资+逻辑+学者+化学+哲学+哲学+娱乐+宇宙+地理+
0.3333333333333333
信任比率:
离间+警惕+利益+了解+贪婪+远离+教育+帮助+纠正+需求+疑虑+进步+决策+思想+了解+
0.35714285714285715
执行比率:
自闭+善良+自恋+平庸+自恋+逆商+优秀+
0.6666666666666666
成功比率:
```

4.6 其他小功能分析例子略.

5. 适用范围

Deta 机器人意识进化系统.
Deta 读心术基础.
Deta 教 育 辅 导 .
Deta 文 本 挖 掘 .
Deta 刑 侦 辅 导 .
Deta 心智训练
Deta 商业用语分析.

6. 注意

注意 1：该正面，褒义，负面，贬义，中性情感语料库有一定比重的表达作者的主观判断，比如思维误差，肯定环境，否定环境，哲学精神论等，如果引起不适，请自定修改词库文件。 如果该情感库对第三方导致任何工程问题，作者不做任何解释和负法律责任。
注意 2: 因为关键字和形谓词模型的应用不确定性，意识和社会形态的溯源问题以及字典理解的误差率，该情感语料库不做任何解释在基于法律与道德的临界线区分应用上。 注意 3: 多语意识场合，该情态库不做任何情形分类评估标准，也不做引导性评估。
注意 4: 该作品免费版本使用权由国际软件研发协议apache-2.0 证书保护. 任何单位任意修改集成使用时请标注 Deta 公司 关键字: “浏阳德塔软件开发有限公司” 或者 “罗瑶光”
注意 5: 当前版本是 10.6.1, 一直在优化中,有任何bug 请直接联系作者.
QQ: 2080315360 (qq: 腾讯)
WECHAT: 15116110525 (WECHAT 微信)
TEL: 15116110525 (tel: 电话号码)
EMAIL: 2080315360@qq.com (email: 邮件地址)

7. 感谢

Deta 的语料库词汇 的 12 国翻译词汇来自有道,百度网的一个词一个词翻译.
Deta 的语料库词汇 的词性词汇来自复旦大学的开源翻译软件一个词一个词标注. Deta 项目设计 采用 Mind Master 软件.
Deta 项目研发 采用 Eclipse IDE 软件. Deta 项目测试 采用 JUNIT API 软件. Deta 项目作品 主要采用 JAVA JDK8+.
Deta 项目语义认知思维能力来自作者学习长达 16 年由中国人民教育出版社出版的国学语文教材. 同时感谢 Lucene 为作者研究提供了启蒙基础.(当时(2009)中科院基于 lucene 内核写中文插件 ,在此标注). 作者长期使用windows 操作系统开发, 电脑装360 杀毒软件保证了8 个月的高效研发环境. 感谢 github 备份, 节省了作者大量的存储硬盘,同时方便 查阅, 逻辑 的鼠标键盘, fhilips 32 寸步屏幕 给作者 提供了迅捷 的输入输出 便利 . 当然 电信的网络, 花生壳的穿透, 老爸,老妈, 都要感谢的.

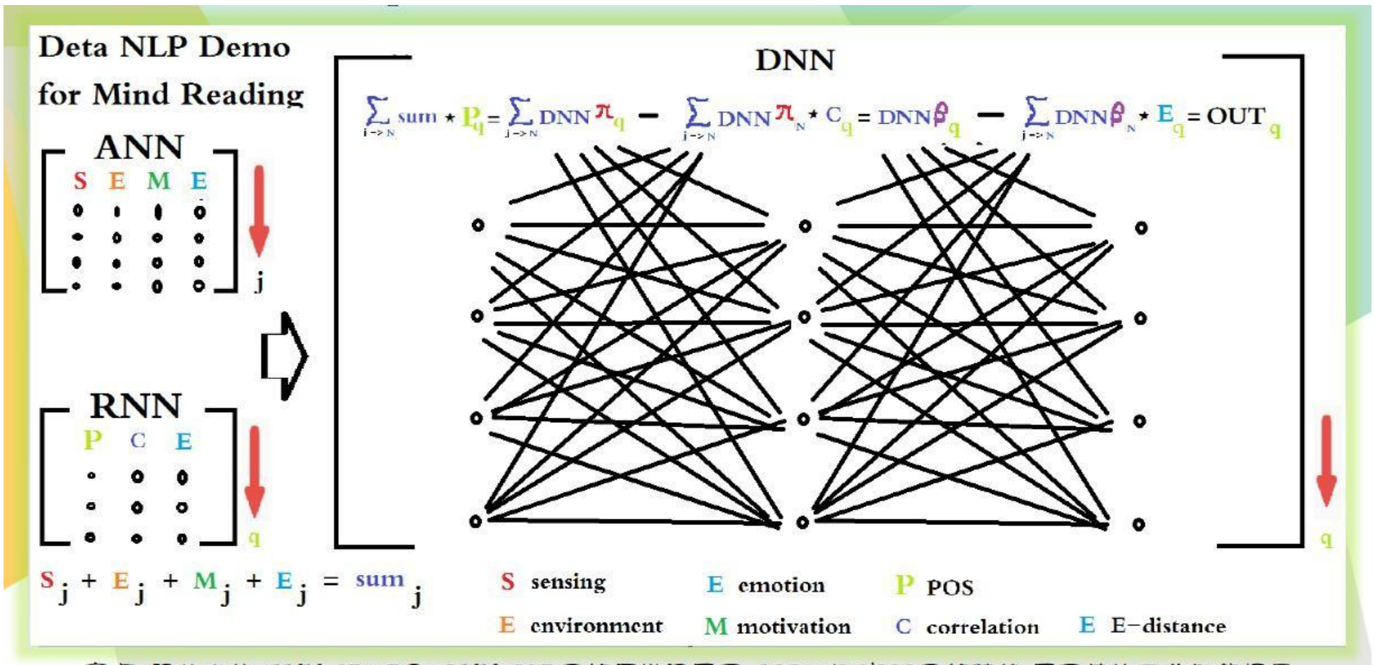
8 研发需要清单

8.1 Java 编辑器.
8.2 Jdk8+. Java 虚拟机运行环境.
8.3 Junit 测试包.
8.4 一台连网的电脑.

德塔读心术词汇重心图算法思想手稿 20190908

罗瑶光

德塔的读心术强调的是增强文章的快速阅读理解，之前我有一张图片发布，我现在列出来如下：



亮点：该公式的 ANN-SEME 和 RNN-PCE 子核同样适用于 AOPM 与 VPCS 子核替换 用于其他工业智能场景

为了方便大家的工程应用，我组织下简单的文字来进行描述下。
从上图。如果有一定经验的数据算法工程师是很容易理解的。如果是新手也不要着急，因为真正问题只是概念描述　的问题。

Deta　的DNN　是一个前序比对累增积分过程的内核算法。需要做这个算法，必要条件是　ANN 的最终运算集合以及 RNN 的卷积内核参照。ANN 是比较基础的东西，基础归基础，应用领域非常强势，2 维的数据永远离不开他。通过 ANN 的计算，我们在处理文章的词汇计算中可以得到一些通用的信息集合，比如文章的敏感度，意识，作者的精神状态，动机，作者当时的多语言环境因素等等，为什么可以得到？原因是比较通俗易懂的，因为褒义，贬义统计，文章的不同的词性的比例，和词汇的转义猜测，和名词的分类引申，这些基础都是非常简单的信息进行普通处理。

RNN 的内核矩阵就麻烦点了。DETA 的 RNN 内核矩阵主要是三个维度：词性的统计值，相同词汇的频率已经在文章中出现的欧几里得距离重心，斜率关联等等，这里需要严谨的算法公式来推导出内核。
有了 ANN 的最终数据集合 和 RNN 的卷积核，我们就可以做 CNN 轮询了 DETA 的DNN 计算定义就是基于德塔的Ann 矩阵数据得到最终1 维数列比，然后进行德塔的RNN 内核做 卷积处理 的3 层深度前序累增积分概率比 CNN 轮循运算。（为了追求更高的质量和精度，小伙伴可以自由改写我的作品思想源码，增加更多的维度皆可。永久开源，别担心著作权问题，有些作品我赠送版权，以后赠予对象如有进行出版社出版，相关文字和内容的引用就要注　意了。）

上面介绍的是ANN，RNN，　CNN 关于公式，环境，原理和初始过程，关于DETA DNN 的计算算法在图片中已经列出来了。我就不多介绍了，
这个算法的相关实现代码的核心部分地址如下：
https://github.com/yaoguangluo/Data_Processor/blob/master/DP/NLPProcessor/DETA_DNN.java

研发源码

```
package org.tinos.view.stable; import java.util.HashMap; import java.util.Map;
        public interface StableMaps{
                public static final Map<String, String> fuCi = new HashMap<>();
                public static final Map<String, String> dongCi= new HashMap<>();
                public static final Map<String, String> liangCi= new HashMap<>();
                public static final Map<String, String> lianCi= new HashMap<>();
                public static final Map<String, String> baDongCi= new HashMap<>();
                public static final Map<String, String> xianDingCi= new HashMap<>();
                public static final Map<String, String> mingCi= new HashMap<>();
                public static final Map<String, String> daiCi= new HashMap<>();
                public static final Map<String, String> jieCi= new HashMap<>();
                public static final Map<String, String> xingRongCi= new HashMap<>();
                public static final Map<String, String> zhuCi= new HashMap<>();
                public static final Map<String, String> weiCi= new HashMap<>();
                public static final Map<String, String> shengLueCi= new HashMap<>();
                public static final Map<String, String> qingTaiCi= new HashMap<>();
                public static final Map<String, String> xingWeiCi= new HashMap<>();
                public static final Map<String, String> shiTaiCi= new HashMap<>();
                public static final Map<String, String> dingMingCi= new HashMap<>();
                public static final Map<String, String> CiOne= new HashMap<>();
                public static final Map<String, String> CiTwo= new HashMap<>();
                public static final Map<String, String> CiThree= new HashMap<>();
                public static final Map<String, String> CiFour= new HashMap<>();
        }
        ---------------------------------------------------------------------------------------------------------------------------
        package org.tinos.view.stable;
        public interface StableData {
               public static final String NLP_CI_MING= "名词";
               public static final String NLP_CI_BA_DONG= "把动词";
               public static final String NLP_CI_DAI= "代词";
               public static final String NLP_CI_DONG= "动词";
               public static final String NLP_CI_DONG_MING= "动名词";
               public static final String NLP_CI_FU= " 副 词 ";
               public static final String NLP_CI_JIE= " 介 词 ";
               public static final String NLP_CI_LIANG= "量词";
               public static final String NLP_CI_SHI_TAI= "时态词";
               public static final String NLP_CI_LIAN= "连词";
               public static final String NLP_CI_QING_TAI= "情态词";
               public static final String NLP_CI_WEI= "谓词";
               public static final String NLP_CI_XING_RONG= "形容词";
               public static final String NLP_CI_XING_WEI= "形谓词";
               public static final String NLP_CI_ZHU= "助词";
               public static final String NLP_CI_SHENG_LUE= "省略词";
               public static final String NLP_CI_XIAN_DING= "限定词";
               public static final String NLP_CI_DING_MING = "定名词";
public static final String NLP_FU_SHU= "复数";
public static final String NLP_ZI_MING= "名";
public static final String NLP_ZI_DONG= "动";
public static final String NLP_ZI_XING= "形";
public static final String NLP_ZI_FU= " 副 ";
public static final String NLP_ZI_WEI= " 谓 ";
public static final String NLP_ZI_JIE= " 介 ";
public static final String NLP_ZI_DAI= " 代 ";
public static final String NLP_ZI_ 复 = " 复 ";
public static final String NLP_ZI_ 单 = " 单 ";
public static final String NLP_ZI_一= "一";
public static final String NLP_HAVE_HAS= "have(has)";
public static final String NLP_HAS= "has";
public static final String NLP_HAVE= "have";
public static final String NLP_ZI_ZAI= "在";
public static final String NLP_SYMBO_SLASH= "/";
public static final String NLP_ZI_ZHONG= " 中 ";
public static final String NLP_ENGLISH_OF= "of";
public static final String NLP_ENGLISH_S= "s";
public static final String NLP_ENGLISH_ES= "es";
public static final String NLP_ENGLISH_ING= "ing";
public static final String NLP_ENGLISH_STATUS= "status";
public static final String NLP_ENGLISH_THE= "the";
public static final String NLP_NULL= "null";
public static final String NLP_DOT= ",";
public static final String NLP_SPASE_REP= "\\s+";
public static final char NLP_CHAR_E= 'e';
public static final char NLP_CHAR_H= 'h';
public static final char NLP_CHAR_S= 's';
public static final int INT_ERROR= -1;
public static final int INT_RIGHT= 1;
public static final int INT_ZERO= 0;
public static final int INT_ONE= 1;
public static final int INT_TWO= 2;
public static final int INT_THREE= 3;
public static final int INT_FOUR= 4;
public static final int INT_FIVE= 5;
public static final int INT_SIX= 6;
public static final int INT_SEVEN= 7;
public static final int INT_TEN= 10;
public static final int INT_EIGHT= 8;
public static final int INT_NINE= 9;
public static final int INT_ELEVEN= 11;
public static final int INT_TWELVE= 12;
public static final int INT_THIRTEEN= 13;
public static final intINT_FOURTEEN= 14;
public static final int INT_NINTY= 90;
```

```
        public static final int INT_NINTY_SEVEN= 97;
        public static final int INT_ONE_TWO_EIGHT= 128;
        public static final int INT_TEN_SOUTHANDS= 10000;
        public static final int INT_ONE_TWO_TWO= 122;
        public static final int INT_SIXTEEN= 16;
        public static final int INT_SIXTY_FOUR= 64;
        public static final String UNLIKELY_ARG_TYPE= "unlikely-arg-type";
        public static final String RAW_TYPES= "rawtypes";
        public static final String EMPTY_STRING= "";
        public static final String SPACE_STRING= " ";
        public static final String SPACE_STRING_DISTINCTION= " ";
        public static final String UNCHECKED= "unchecked";
        public static final String GBK_STRING= "GBK";
        public static final String UTF8_STRING= "UTF8";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_CN= "poscc.lyg";
        public static final String WORDS_SOURSE_LINK_POS_EN_TO_CN= "posec.lyg";
        public static final String WORDS_SOURSE_LINK_POS_EN_TO_EN= "posee.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_FN= "poscf.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_KO= "posck.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_JP= "poscj.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_GM= "poscg.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_SP= "poscs.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_RS= "poscr.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_AB= "posca.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_PY= "pospy.lyg";
        public static final String WORDS_SOURSE_LINK_POS_CN_TO_TT ="postt.lyg";
        public static final String WORDS_SOURSE_LINK_POS_NEGATIVE= "posNegative.lyg";
        public static final String WORDS_SOURSE_LINK_POS_POSITIVE= "posPositive.lyg";
        public static final String WORDS_SOURSE_LINK_MOTIVATION= "motivation.lyg";
        public static final String WORDS_SOURSE_LINK_TRENDING= "trend.lyg";
        public static final String WORDS_SOURSE_LINK_PREDICTION= "prediction.lyg";
        public static final String WORDS_SOURSE_LINK_DISTINCTION= "distinction.lyg";
        public static final String WORDS_SOURSE_LINK_EN_TO_CN= "ec.lyg";
        public static final String WORDS_SOURSE_LINK_CN_TO_EN= "ce.lyg";
        public static final String NUMBERS= "1234567890";
}
package org.tinos.view.obj;
public class WordFrequency {
public String getPOS() { return POS;

 }
public void setPOS(String POS) {
         this.POS = POS;
 }public String getWord() {
         return word;
 }
public void setWord(String word) {
         this.word = word;
 }
```

```
public int getFrequency() {
            return frequency;
 }
public void setFrequency(int frequency) {
            this.frequency = frequency;
 }
public WordFrequency getLeft() {
            return left;
 }
public void setLeft(WordFrequency left) {
            this.left = left;
 }
public WordFrequency getRight() {
            return right;
 }
public void setRight(WordFrequency right) {
            this.right = right;
 }
 private String word;
 private String POS;
 private int frequency;
 private WordFrequency left;
 private WordFrequency right;
}
----------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.obj;
public class StringOBJ{
public String getEnglishString() {
            return englishString;
 }
public void setEnglishString(String englishString) {
            this.englishString = englishString;
 }
public String getChineseString() {return chineseString;
 }
public void setChineseString(String chineseString) {
            this.chineseString = chineseString;
 }
 private String englishString;
 private String chineseString;
}
----------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.obj;
public abstract class HMMNode {
public String getVb() {
            return vb;
 }
public void setVb(String vb) {
```

```
            this.vb = vb;
 }
 private String vb;
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.obj;
import java.util.Map;
public class FMHMMPOS extends HMMNode {
public Map<String, Integer> getNext() {
            return next;
 }
public void setNext(Map<String, Integer> next) {
            this.next = next;
 }
 private Map<String, Integer> next;
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.obj;
import java.util.Map;
public class FMHMMNode extends HMMNode {
public Map<String, Integer> getNext() {
            return next;
 }
public void setNext(Map<String, Integer> next) {this.next = next;
 }
 private Map<String, Integer> next;
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.exception;
import java.io.IOException;
public class PaserException extends IOException {
private static final long serialVersionUID = -6219387333580082136L;
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.test;
import java.util.Date;
public class TimeCheck{
    public long before;
    public long now;
    public void begin(){
          System.out.println("start: " + (new Date())); before = System.currentTimeMillis();
    }
    public void end(){
            now = System.currentTimeMillis();
    }
    public void duration(){
            long du=now-before;
            System.out.println("锟斤拷时: " + du + "  锟斤拷锟斤拷");
```

```
      }
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.view.obj; @SuppressWarnings("unused")
public class Verbal{
public String getChinese() {
             return chinese;
 }
public void setChinese(String chinese) {
             this.chinese = chinese;
 }
public String getEnglish() {
             return english;
 }
public void setEnglish(String english) {
             this.english = english;
 }
public String getPartOfSpeech() {
             return partOfSpeech;}
public void setPartOfSpeech(String partOfSpeech) {
             this.partOfSpeech = partOfSpeech;
 }
public String getExplain() {
             return explain;
 }
public void setExplain(String explain) {
             this.explain = explain;
 }
public Verbal getNext() {
             return next;
 }
public void setNext(Verbal next) {
             this.next = next;
 }
public Verbal getPrev() {
             return prev;
 }
public void setPrev(Verbal prev) {
             this.prev = prev;
 }
private String chinese;
private String japanese;
private String  korea;
private String russian;
private String arabic;
private String french;
private String german;
private String spanish;
```

```
          private String pinyin;
          private String english;
          private String partOfSpeech;
          private String explain; private Verbal next;
           private Verbal prev;
}
package org.tinos.test;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import org.tinos.view.stable.StableData;
@SuppressWarnings("unused") public class testCharAt {
public static void main(String[] args) throws InterruptedException, IOException {
                              }}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.test;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class MakeImage {
        public static void main(String[] args) throws IOException {
                    MakeImage makeImage= new MakeImage();
                    int height= 600;
                    int weight= 800;
                    BufferedImage image= new BufferedImage(weight, height, BufferedImage.TYPE_INT_RGB);
                    Graphics2D g= image.createGraphics();
                    g.setColor(Color.white);
                    g.setFont(g.getFont());
                    String prePrint= "测试文字";
                    Font font= new Font(prePrint,20, 20);
                    int stringW= g.getFontMetrics().stringWidth(prePrint);
                    int stringH= g.getFontMetrics().getHeight();
                    System.out.println(stringW+ ":"+ stringH);
                    int positionX= 10;
                    int positionY= 20;
                    makeImage.weightPrint(prePrint, g, stringW, stringH, positionX, positionY, font);
                    makeImage.heightPrint(prePrint,g, stringW, stringH, positionX, positionY, font);
                    String pathBin= "C:\\Users\\Administrator\\Desktop\\书\\images\\output.jpg";
                    //output path
                    File outputBin= new File(pathBin);
                    ImageIO.write(image, "jpeg", outputBin);
          }
        public void weightPrint(String prePrint, Graphics2D g, int stringW, int stringH, int positionX, int positionY, Font font) {
                    g.setFont(font);
                    g.drawString(prePrint, positionX, positionY);
          }
        public void heightPrint(String prePrint, Graphics2D g, int stringW, int stringH, int positionX, int positionY, Font font) {
                    g.setFont(font);
                    g.drawString(prePrint, positionX, positionY);
          }}
package org.tinos.test;
import java.util.HashMap;
import java.util.Map;
public class EffectionDemo{
        public static void main(String[] argv) {
                    TimeCheck timeCheck= new TimeCheck();
                    //titan sets System.out.println(1);
```

```
Map<String,String> map= new HashMap<>();
        for(int i=0;i<100000;i++) {
        Map.put(""+i, ""+i);
        }
        System.out.println(2);
        //subset
Map<String,String> map1= new HashMap<>();
        for(int i=0;i<30000;i++) {
        Map1.put(""+i, ""+i);
        }
        System.out.println(3);
Map<String,String> map2= new HashMap<>();
        for(int i=0;i<30000;i++) {
        Map2.put(""+i, ""+i);
        }
        System.out.println(4);
Map<String,String> map3= new HashMap<>();
        for(int i=0;i<30000;i++) {
        Map3.put(""+i, ""+i);
        }
        // System.out.println(5);
        //
Map<String,String> map4= new HashMap<>();
        //
        for(int i=0;
i<20000;i++) {
                //
        Map4.put(""+i, ""+i);
                //
        }
        // System.out.println(6);
        //
Map<String,String> map5= new HashMap<>();
        //
        for(int i=0;i<20000;i++) {
                //
        Map5.put(""+i, ""+i);
                //
        }
        System.out.println(7);
        //time check timeCheck.begin();
        function1(map);
        timeCheck.end();
        timeCheck.duration();
        // timeCheck.begin();
        function2(map1,map2,map3);
        timeCheck.end();
        timeCheck.duration();
}
private static void function2(Map<String, String> map1, Map<String, String> map2, Map<String, String> map3) {
        Here:
                for(int i=0;i<100000;i++) {
                        int digits= (int)(Math.random()*30000);
                        if(map1.containsKey(digits+"")) {
                                continue Here;
                        }
                        if(map2.containsKey(digits+"")) {
                                continue Here;
                        }
                        if(map3.containsKey(digits+"")) {
                                continue Here;
                        }
                }
}
private static void function1(Map<String, String> map) {
        Here1:
                for(int i=0;i<100000;i++) {
```

```
				int digits= (int)(Math.random()*30000);
				if(map.containsKey(digits+"")) {
					continue Here1;
				}
			}
	}}
```

```
package org.tinos.test;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.view.obj.WordFrequency;
import static java.lang.System.*;
@SuppressWarnings("unused") public class DemoPOSMedcine {
public static void main(String[] args) throws IOException {
		Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
		analyzer.init();
	Map<String, String> nlp= analyzer.getPosCnToCn();
	List<String> sets= new ArrayList<>();
		String[] ss= new String[1];
		String[] ss1= new String[1];
		ss[0]= "";
		ss1[0]= "";
		//ss1[0]= "、益母 、结婚益母";
		for (int i= 0;i < ss.length;i++) {
			System.out.println("超级变态复杂病句-->"+ ss[i]);
		Map<String, WordFrequency> map= analyzer.parserStringByReturnFrequencyMap( ss[0]);
			out.print("分析处理真实结果-->");iterator<String> it= map.keySet().iterator();
			while(it.hasNext()) {
				out.println(it.next());
			}
		}}}
```

```
package org.tinos.test;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import static java.lang.System.*;
public class DemoPOSforSpecial {
public static void main(String[] args) throws IOException {
		Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
		analyzer.initMixed();
	Map<String, String> nlp= analyzer.getPosCnToCn();
	List<String> sets= new ArrayList<>();
		String[] ss= new String[1];
		String[] ss1= new String[1];
		ss[0]= "存在这是非常常是非常愚蠢的为主要求知不断的提高自身的知识的不锻炼改变化나는일을준비하고있다";
		ss1[0]= "나는 일을 준비 하고 있다 ";

		for (int i= 0;i< ss.length;i++) {
			System.out.println("超级变态复杂病句-->"+ ss[i]);
			sets= analyzer.parserMixedString(ss[i].replace(" ", " "));
			//词性分析
			out.print("分析处理真实结果-->");
			for (int j= 0;j< sets.size();j++) {
				if (!sets.get(j).replaceAll("\\s+", "").equals("")) {
					out.print(sets.get(j)+ " ");
				}
			}
```

```
                    out.println();
                    out.println("期望得到分词效果-->"+ ss1[i]);
                    for (int k= 0;k< sets.size();k++) {
                    if (!sets.get(k).replaceAll("\\s+", "").equals("")) {
                              nlp.get(sets.get(k));
                              out.println(sets.get(k)+ "/"+ nlp.get(sets.get(k))+ " ");
                    }
          }
          out.println("");
}}}
```

```
package org.tinos.test;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import static java.lang.System.*;
public class DemoPOS {
public static void main(String[] args) throws IOException {
                    Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
                    analyzer.init();
          Map<String, String> nlp= analyzer.getPosCnToCn();
          List<String> sets= new ArrayList<>();
                    String[] ss= new String[37];
                    String[] ss1= new String[37];
                    ss[0]= "";
                    //下面一些测句子来自 github 上 国际 官方用于分词的准确率评估例句, 不属于函数逻辑源码实体.
                    ss[1]= "海南方向逃跑";
                    ss[2]= "他说的确实在理";
                    ss[3]= "";
                    ss[4]= "";
                    ss[5]= "提高产品质量";
                    ss[6]= "中外科学名著";
                    ss[7]= "北京大学生前来应聘";
                    ss[8]= "为人民服务";
                    ss[9]= "独立自主和平等互利的原则";
                    ss[10]= "为人民办公益";
                    ss[11]= "这事的确定不下来";
                    ss[12]= "这扇门把手坏了";
                    ss[13]= "他把手抬起来";
                    ss[14]= "学生会宣传部";
                    ss[15]= "学生会主动完成作业";
                    ss[16]= "学生会游戏";
                    ss[17]= "研究生活水平";
                    ss[18]= "中国有企业";
                    ss[19]= "我爱美国手球";
                    ss[20]= "中国喜欢狗";
                    ss[21]= "中国热爱狗";
                    ss[22]= "王军虎去广州了";
                    ss[23]= "王军虎头虎脑的";
                    ss[24]= "将军任命了一名中将";
                    ss[25]= "产量三年中将增长两倍";
                    ss[26]= "";
                    ss[27]= "我来到北京清华大学";
                    ss1[0]= "";
                    ss1[1]= "海 南 方向 逃跑";
                    ss1[2]= "他 说 的 确实 在理";
                    ss1[3]= "";
                    ss1[4]= "";
                    ss1[5]= "提高 产品 质量";
                    ss1[6]= "中外 科学 名著";
```

```
		ss1[7]= "北京 大学生 前 来 应聘";
		ss1[8]= "为 人民 服务";
		ss1[9]= "独立 自主 和 平等 互利 的 原则";
		ss1[10]= "为 人民 办公益";
		ss1[11]= "这事 的确 定 不 下来";
		ss1[12]= "这 扇 门 把手 坏 了";
		ss1[13]= " 他 把 手 抬 起 来 ";
		ss1[14]= "学生会 宣传 部";
		ss1[15]= "学生 会 主动 完成 作业";
		ss1[16]= "学生会 游戏";
		ss1[17]= "研究 生活 水平";
		ss1[18]= "中国 有 企业";
		ss1[19]= "我 爱 美国 手球";
		ss1[20]= "";
		ss1[21]= "";
		ss1[22]= " 王 军 虎 去 广 州 了 ";
		ss1[23]= " 王 军 虎 头 虎 脑 的 ";
		ss1[24]= "将军 任命 了 一名 中将";
		ss1[25]= "产量 三 年 中 将 增长 两倍";
		ss1[26]= "";
		ss1[27]= "我 来到 北京 清华 大学";
		ss[28]= "";
		ss1[28]= "";
		ss[29]= "";
		ss1[29]= "";
		ss[30]= " ";
		ss1[30]= " ";
		ss[31]= " ";
		ss1[31]= "";
		ss[32]= " ";
		ss1[32]= " ";
		ss[33]= "老人家身体不错";
		ss[34]= "老人家中很干净";
		ss1[33]= "老 人家 身体 不错";
		ss1[34]= "老人 家 中 很 干净";
		ss[35]= "版权归属做出回应";
		ss[36]= "有用户发现";
		ss1[35]= "版权 归属 做 出 回应";
		ss1[36]= "  有 用户 发现";
		for (int i= 0;i < ss.length;i++) {
		System.out.println("超级变态复杂病句-->"+ ss[i]);
		sets= analyzer.parserString(ss[i].replace(" ", ""));
		//词性分析
		out.print("分析处理真实结果-->");
		for (int j= 0;j < sets.size();j++) {
			if (!sets.get(j).replaceAll("\\s+", "").equals("")) {
				out.print(sets.get(j)+ " ");
			}
		}
		out.println();
		out.println("期望得到分词效果-->"+ ss1[i]);
		for (int k= 0;k < sets.size();k++) {
			if (!sets.get(k).replaceAll("\\s+", "").equals("")) {
				nlp.get(sets.get(k));
				out.println(sets.get(k)+ "/"+ nlp.get(sets.get(k))+ " ");
			}
		}
		out.println("");
	}
}}
```

```
package org.tinos.test;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.view.obj.WordFrequency;
public class DemoEX {
        int a=0;int b=0;int c=0;int d=0;int e=0;int f=0;int g=0;
        @SuppressWarnings("unused")public static void main(String[] args) throws IOException {
                Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
                //analyzer.init();
                analyzer.initMixed();
        Map<String, String> pos= analyzer.getPosCnToCn();
        List<String> sets= new ArrayList<>();
        Map<String, WordFrequency> seta= new ConcurrentHashMap<>();
                TimeCheck t= new TimeCheck();
                String ss= "科学的发展是一种市场和社会需求的传承，每一位获得诺贝尔奖的科学家";
                //32 字
                DemoEX demoEX=new DemoEX();
                t.begin();
                for (int i= 0;i < 5000000;i++) {
                        //重复 500 万次数 相当于处理 1.6 亿字 耗费 时 7.280 秒
                        sets= analyzer.parserMixedString(ss);
                }
                t.end();
                t.duration();
                System.out.println(StableCount.a1);
                System.out.println(StableCount.a2);
                System.out.println(StableCount.a3);
                System.out.println(StableCount.a4);
                //// System.out.println(StableCount.a5);
                //// System.out.println(StableCount.a6);
                //// System.out.println(StableCount.a7);
                //// System.out.println(StableCount.a8);
                //// System.out.println(StableCount.a9);
                //// System.out.println(StableCount.a10);
                //// System.out.println(StableCount.a11);
                //// System.out.println(StableCount.a12);
                //
                for (int i= 0;i < sets.size();i++) {
                //
                        if (sets.get(i) != null) {
                        //
                        System.out.print(sets.get(i)+ " ");
                        //
                }
                //
                }
                ////
                for (int i= 0;i < sets.size();i++) {
                ////
                        if (!sets.get(i).equals("")) {
                        ////
                        System.out.print(sets.get(i)+ " ");
                        ////
                }
                ////
                }
                // System.out.println("");
                //// t.duration();
                // System.out.println("");
                //
                System.out.println("词性分析-->");
                //// t.begin();
                //
                for (int j= 0;j < 1;j++) {
```

```
                //
                        for (int i= 0;i < sets.size();i++) {
                        //
                                if (!sets.get(i).replaceAll("\\s+", "").equals("")) {
                        //
                                        System.out.print(sets.get(i)+ "/"+ pos.get(sets.get(i))+ "");
                        //
                                }
                        //
                        }//
                }
                //// t.end();
                // System.out.println("");
                //// t.duration();
                // System.out.println("");
                //
                System.out.println("词频分析-->");
                //// t.begin();
                //
        Map<Integer, WordFrequency> fwa= analyzer.getWordFrequencyByReturnSortMap(sets);
                //// t.end();
                //
                for (int i= fwa.size() - 1;i >= 0;i--) {
                //
                        System.out.print(fwa.get(i).getWord()+ ":"+ fwa.get(i).getFrequency()+ "");
                //
                }
                // System.out.println("");
                //// t.duration();
        }}
-------------------------------------------------------------------------------------------------------------------------------
package org.tinos.test;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import java.io.IOException;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
                @SuppressWarnings("unused") public class DemoCogs {
                        static List<List<String>> sets;
                        static String ss;
                        static int c= 0;
public static void main(String[] args) throws IOException, InterruptedException {
                Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
                //Analyzer analyzer= new FastAnalyzerImp();
                //Analyzer analyzer= new PrettyAnalyzerImp();
                //Analyzer analyzer= new BaseAnalyzerImp();
                //Analyzer analyzer= new ScoreAnalyzerImp();
                analyzer.init();
                sets= new CopyOnWriteArrayList<>();
                String ss= "从容易开始从容易知";
                System.out.println("");
                ExecutorService executorService= Executors.newFixedThreadPool(1);
                for (int i= 0;i < 100;i++) {
                executorService.submit(new TaskWithResult(i, analyzer, ss));
                } while (sets.size() < 100) {
                Thread.sleep(300);
                }
                int j= 0;
                for (int i= 0;i < sets.get(0).size();i++) {
                System.out.print(sets.get(0).get(i)+ " | ");
                j++;
        if (j > 15) {
                        j= 0;
                        System.out.println("");
```

```
                    }
                    }
                              }
public static class TaskWithResult implements Callable<String> {
          private int id;
          private Analyzer analyzer;
          private String sss;
public TaskWithResult(int id, Analyzer analyzer, String sss) {
                    this.id= id;
                    this.analyzer= analyzer;
                    this.sss= sss;
                    }
public String call() throws Exception {
          List<String> te= analyzer.parserString(sss);
                    sets.add(te);
                    return null;
                    }
                              }}
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.ortho.fhmm;
import java.util.Map;
import org.tinos.view.stable.StableData;
public interface FMHMMList extends FHMMList {
                    @SuppressWarnings(StableData.RAW_TYPES) Map<Integer, Map> getRoot();
          }
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.ortho.fhmm;
import org.tinos.view.obj.FMHMMNode;
//词汇翻译系统
import java.io.IOException;
import java.util.List;
import java.util.Map;
public interface FHMMList {
          void index() throws IOException;
          void indexMixed() throws IOException;
          void indexPosEnToCn() throws IOException;
          void indexPosEnToEn() throws IOException;
          void indexEnToCn() throws  IOException;
          void indexCnToEn() throws  IOException;
          void indexFullEnToCn() throws IOException;
          void indexFullCnToEn() throws IOException;
          void indexFullCnToJp() throws IOException;
          void indexFullCnToRs() throws IOException;
          void indexFullCnToAb() throws IOException;
          void indexFullCnToFn() throws IOException;
          void indexFullCnToGm() throws IOException;
          void indexFullCnToKo() throws IOException;
          void indexFullCnToSp() throws IOException;
          void indexFullCnToPy() throws IOException;
          void indexFullNegative() throws IOException;
          void indexFullPositive() throws IOException;
          Map<Long, FMHMMNode> getMap();
          Map<Long, FMHMMNode>[] getMaps();
          Map<String, String> getPosEnToCn();
          Map<String, String> getPosEnToEn();
          Map<String, String> getPosCnToCn();
          Map<String, String> getEnToCn();
          Map<String, String> getCnToEn();
          Map<String, String> getFullEnToCn();
          Map<String, String> getFullCnToEn();
          Map<String, String> getFullCnToJp();
          Map<String, String> getFullCnToRs();
          Map<String, String> getFullCnToAb();
          Map<String, String> getFullCnToFn();
          Map<String, String> getFullCnToGm();
          Map<String, String> getFullCnToKo();
          Map<String, String> getFullCnToSp();
```

```
		Map<String, String> getFullCnToPy();
		Map<String, String> getFullNegative();
		Map<String, String> getFullPositive();
		List<String> englishStringToWordsList(String string);
		Map<Long, Map<String, String>> getWordsForests();
		void studyNewPos(String string, String posStudy);
		Map<String, String> getStudyPos();
}
-------------------------------------------------------------------------------------------------------------------------------
package org.tinos.ortho.fhmm.imp;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentHashMap;
import org.tinos.view.stable.StableData;
import timeProcessor.TimeCheck;
public class PillowsSet{
public long index;
public boolean entry= false;
public Object object;
public PillowsSet smallOrder;
public PillowsSet largeOrder;
public PillowsSet preSmallOrder;
public PillowsSet preLargeOrder;
public void arrangePillow(long index, Object object, int pillows, int depth, int currentDepth) throws CloneNotSupportedException {
		if(null== this.object) {
				this.index= index;
				this.object= object;
				return;
		}
		if(index< this.index) {
				if(null== smallOrder) {
				smallOrder= new PillowsSet();
				smallOrder.preLargeOrder= this;
				}
				smallOrder.arrangePillow(index, object, pillows, depth, currentDepth+ StableData.INT_ONE);
		}
		if(index> this.index) {
				if(null== largeOrder) {
						largeOrder= new PillowsSet();
						largeOrder.preSmallOrder= this;
				}
				largeOrder.arrangePillow(index, object, pillows, depth, currentDepth+ StableData.INT_ONE);
		}}
public Object getPillow(long index) {
		if(index== this.index) {
				return object;
		}
		if(index< this.index) {
				return smallOrder.getPillow(index);
		}
		if(index> this.index ) {
				return largeOrder.getPillow(index);
		}
		return null;
}
public void superBalance() {}
public void show() {
		entry= true;
		if(smallOrder!= null&& !smallOrder.entry) {
				smallOrder.show();
		}
		if(largeOrder!= null&& !largeOrder.entry) {
				largeOrder.show();
		}}
```

```
public static void main(String[] argv) throws CloneNotSupportedException {
        int pillows= StableData.INT_ZERO;
        int depth= pillows >> StableData.INT_ONE;
        int currentDepth= StableData.INT_ZERO;
        PillowsSet pillowsMap= new PillowsSet();
        for(int i=StableData.INT_ZERO;i<5000;i++) {
                pillowsMap.arrangePillow(i, i, pillows++, depth, currentDepth);
        }
        pillowsMap.fixEntry();
        pillowsMap.show();
        //
        TimeCheck timecheck= new TimeCheck();
        // timecheck.begin();
        //
        for(int i=0;i<100000;i++) {
        //// pillowsMap.getPillow(100);
        //}
        // timecheck.end();
        // timecheck.duration();
        System.out.println(pillowsMap.getPillow(100));
        Map<Long, Object> map= new HashMap<>();
        for(long i=0;i<5000000;i++) {
                Map.put(i, i);
        }
        long v=1000;
        TimeCheck timecheck= new TimeCheck();
        timecheck.begin();
        for(int i=0;i<5000000;i++) {
                Map.get(v);
        }
        timecheck.end();
        timecheck.duration();
        System.out.println(map.get(v));
        //c map= new ConcurrentHashMap<>();
        for(long i=0;i<5000000;i++) {
                Map.put(i, i);
        } v=1000;
        timecheck= new TimeCheck();
        timecheck.begin();
        for(int i=0;i<5000000;i++) {
                Map.get(v);
        }
        timecheck.end();
        timecheck.duration();
        System.out.println(map.get(v));
        //map compare
        Map= new LinkedHashMap<>();
        for(long i=0;i<5000000;i++) {
                Map.put(i, i);
        } v=1000;
        timecheck.begin();
        for(int i=0;i<5000000;i++) {
                Map.get(v);
        }
        timecheck.end();
        timecheck.duration();
        System.out.println(map.get(v));
        //
        Hashtable<Long, Object> table= new Hashtable<>();
        for(long i=0;i<5000000;i++) {
                        table.put(i, i);
        } v=1000;
        timecheck.begin();
        for(int i=0;i<5000000;i++) {
                table.get(v);
        }
        timecheck.end();
```

```
            timecheck.duration();
            System.out.println(table.get(v));
            //
            TreeMap<Long, Object> tree= new TreeMap<>();
            for(long i=0;i<5000000;i++) {
                        tree.put(i, i);
            }
            v=1000;
            timecheck.begin();
            for(int i=0;i<5000000;i++) {
                        tree.get(v);
            }
            timecheck.end();
            timecheck.duration();
            System.out.println(tree.get(v));
}
private void fixEntry() {
            int small= 0;
            int large= 0;
            if(largeOrder == null && preLargeOrder!=null) {
                        largeOrder= preLargeOrder;if(largeOrder.preLargeOrder!=null) {
                                    preLargeOrder= largeOrder.preLargeOrder;
                                    preLargeOrder.smallOrder= this;
                        }else if(largeOrder.preSmallOrder!=null) {
                                    preSmallOrder= largeOrder.preSmallOrder;
                                    preSmallOrder.largeOrder= this;
                        }
                                    largeOrder.smallOrder= null;
                                    large=1;
                        }
            if(smallOrder == null && preSmallOrder!=null) {
                        smallOrder= preSmallOrder;if(smallOrder.preSmallOrder != null) {
                                    preSmallOrder= smallOrder.preSmallOrder;
                                    preSmallOrder.largeOrder= this;
                        }else if(smallOrder.preLargeOrder!=null) {
                                    preLargeOrder= smallOrder.preLargeOrder;
                                    preLargeOrder.smallOrder= this;
                        }
                                    smallOrder.largeOrder= null;
                                    small=1;
            }
            if(smallOrder != null&&small==0) {
                        smallOrder.fixEntry();
            }
            if(largeOrder != null&&large==0) {
                        largeOrder.fixEntry();
            }}}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.ortho.fhmm.imp;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
//import java.util.concurrent.HashMap;
import java.util.concurrent.CopyOnWriteArrayList;
import org.tinos.ortho.fhmm.FHMMList;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.obj.FMHMMPOS;
import org.tinos.view.stable.StableData;
import org.tinos.view.stable.StableMaps;
import NLPProcessor.StopSymbol;
```

```
//I will build a collection class for managing this maps. at the next version. @SuppressWarnings("unchecked")
public class FMHMMListOneTimeImp implements FHMMList {
        private Map<String, String> studyPos;
        private Map<String, String> posCnToCn;
        private Map<String, String> posEnToEn;
        private Map<String, String> posEnToCn;
        private Map<String, String> enToCn;
        private Map<String, String> cnToEn;
        private Map<String, String> fullEnToCn;
        private Map<String, String> fullCnToEn;
        private Map<String, String> fullCnToFn;
        private Map<String, String> fullCnToKo;
        private Map<String, String> fullCnToJp;
        private Map<String, String> fullCnToSp;
        private Map<String, String> fullCnToAb;
        private Map<String, String> fullCnToGm;
        private Map<String, String> fullCnToRs;
        private Map<String, String> fullCnToPy;
        private Map<String, String> fullPositive;
        private Map<String, String> fullNegative;
        private List<String> listEn;
        private List<String> listCn;
        private List<String> listFn;
        private List<String> listKo;
        private List<String> listJp;
        private List<String> listSp;
        private List<String> listGm;
        private List<String> listRs;
        private List<String> listAb;
        private List<String> listPy;
        private List<String> listTt;
        public Map<Long, FMHMMPOS> POSHashMap;
        public Map<Long, FMHMMNode> linkedHashMap;
        public Map<Long, FMHMMNode> getMap() {
                return this.linkedHashMap;
        }
        public Map<Long, FMHMMNode>[] getMaps() {
                int segment= this.linkedHashMap.size();int perRatio= segment/ StableData.INT_SIX;
                Map<Long, FMHMMNode>[] maps= new HashMap[StableData.INT_SIX];iterator<Long> iterator=
this.linkedHashMap.keySet().iterator();
                Maps[StableData.INT_ZERO]= new HashMap<>();int index= StableData.INT_ZERO;int count= StableData.INT_ONE;
                while(iterator.hasNext()) {
                        if(StableData.INT_ZERO== count++ % perRatio) {
                                if(index< StableData.INT_FIVE) {
                                        index++;
                                        Maps[index]= new HashMap<>();
                                }
                        }
                        Long key= iterator.next();
                        maps[index].put(key, this.linkedHashMap.get(key));
                }
                return maps;
        }
        public void indexMixed() throws IOException {
                studyPos= new ConcurrentHashMap<>();
                posCnToCn= new ConcurrentHashMap<>();
                linkedHashMap= new ConcurrentHashMap<>();
                ListCn= new CopyOnWriteArrayList<>();
                ListKo= new CopyOnWriteArrayList<>();
                ListJp= new CopyOnWriteArrayList<>();
                ListTt= new CopyOnWriteArrayList<>();
                ListEn= new CopyOnWriteArrayList<>();
                inputStream inputStream=
                getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_CN);
                BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream,
        StableData.UTF8_STRING));
                inputStream inputStreamKorea=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_KO);
                BufferedReader cReaderKorea= new BufferedReader(new InputStreamReader(inputStreamKorea,
        StableData.UTF8_STRING));
                inputStream inputStreamJapan=
                getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_JP);
                BufferedReader cReaderJapan= new BufferedReader(new InputStreamReader(inputStreamJapan,
        StableData.UTF8_STRING));
                inputStream inputStreamTrandition=
                getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_TT);
                BufferedReader cReaderTrandition= new BufferedReader(new          InputStreamReader(inputStreamTrandition,
StableData.UTF8_STRING));
                inputStream inputStreamEnglish=
                getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_EN_TO_EN);
                BufferedReader cReaderEnglish= new BufferedReader(new InputStreamReader(inputStreamEnglish,
        StableData.UTF8_STRING));
                String cInputString;
                String cInputStringKorea;
                String cInputStringJapan;
                String cInputStringTrandition;
                String cInputStringEnglish;
                Here:
                while ((cInputString= cReader.readLine()) != null) {
                cInputStringKorea= cReaderKorea.readLine();
                cInputStringJapan= cReaderJapan.readLine();
                cInputStringTrandition= cReaderTrandition.readLine();
                cInputStringEnglish= cReaderEnglish.readLine();
                listCn.add(cInputString);
                if(null!= cInputStringKorea) {
                        ListKo.add(cInputStringKorea);
                }
                if(null!= cInputStringJapan) {
                        ListJp.add(cInputStringJapan);
                }
                if(null!= cInputStringTrandition) {
                        ListTt.add(cInputStringTrandition);
                }
                if(null!= cInputStringEnglish) {
                        ListEn.add(cInputStringEnglish);
                }
                if(!(!cInputString.replace(StableData.SPACE_STRING,
        StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)&&
        cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
        if(!StableMaps.fuCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_FU)) {
        StableMaps.fuCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                if(null!= cInputStringKorea) {
        StableMaps.fuCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stable
Data.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.fuCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stable
Data.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.fuCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
     if(null!= cInputStringEnglish) {
        StableMaps.fuCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
}
        if(!StableMaps.dongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DONG)) {
        StableMaps.dongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.dongCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.dongCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.dongCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.dongCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.liangCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_LIANG)) {
        StableMaps.liangCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.liangCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
if(null!= cInputStringJapan) {
        StableMaps.liangCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
 if(null!= cInputStringTrandition) {
        StableMaps.liangCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
if(null!= cInputStringEnglish) {
        StableMaps.liangCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.lianCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_LIAN)) {
        StableMaps.lianCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.lianCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stab
leData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.lianCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.lianCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.lianCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.baDongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])  &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_BA_DONG)) {
        StableMaps.baDongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.baDongCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
Stabl eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
```

```
	StableMaps.baDongCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringTrandition) {
	StableMaps.baDongCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.s
plit(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringEnglish) {
	StableMaps.baDongCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}}
	if(!StableMaps.xianDingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XIAN_DING)) {
	StableMaps.xianDingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
		cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
	if(null!= cInputStringKorea) {
	StableMaps.xianDingCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringJapan) {
	StableMaps.xianDingCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringTrandition) {
	StableMaps.xianDingCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.
split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringEnglish) {
	StableMaps.xianDingCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spl
it(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}}
	if(!StableMaps.mingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_MING))
	{			StableMaps.mingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
	if(null!= cInputStringKorea) {
	StableMaps.mingCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringJapan) {
	StableMaps.mingCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringTrandition) {
	StableMaps.mingCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringEnglish) {
	StableMaps.mingCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}}
	if(!StableMaps.daiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DAI)) {
	StableMaps.daiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
		cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
	if(null!= cInputStringKorea) {
	StableMaps.daiCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringJapan) {
	StableMaps.daiCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
	if(null!= cInputStringTrandition) {
	StableMaps.daiCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
		}
```

```
        if(null!= cInputStringEnglish) {
        StableMaps.daiCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.jieCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_JIE)) {
        StableMaps.jieCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.jieCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stable
Data.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.jieCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stable
Data.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.jieCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(S
tableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.jieCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stab
leData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.xingRongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XING_RONG)) {
        StableMaps.xingRongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.xingRongCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spli
t(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.xingRongCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.xingRongCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.
spli t(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.xingRongCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.sp
lit(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.zhuCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_ZHU)) {
        StableMaps.zhuCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.zhuCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.zhuCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.zhuCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.zhuCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.weiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])&&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_WEI)) {
        StableMaps.weiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.weiCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.weiCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Stabl
eDat a.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.weiCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.weiCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(Sta
bleData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.shengLueCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&

        cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_SHENG_LUE))
{               StableMaps.shengLueCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.shengLueCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spli
t(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.shengLueCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.shengLueCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.
split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.shengLueCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.sp
lit(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.qingTaiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_QING_TAI)) {
        StableMaps.qingTaiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea)
                {
        StableMaps.qingTaiCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.qingTaiCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(S
tableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.qingTaiCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.s
plit(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.qingTaiCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.xingWeiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XING_WEI)) {
        StableMaps.xingWeiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.xingWeiCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.xingWeiCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.xingWeiCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.
split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.xingWeiCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spli
t(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.shiTaiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])&&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_SHI_TAI)) {
        StableMaps.shiTaiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.shiTaiCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.shiTaiCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(St
ableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.shiTaiCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spl
it(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish) {
        StableMaps.shiTaiCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split(
StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        if(!StableMaps.dingMingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DING_MING)) {
        StableMaps.dingMingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        if(null!= cInputStringKorea) {
        StableMaps.dingMingCi.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.spli
t(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringJapan) {
        StableMaps.dingMingCi.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.split
(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringTrandition) {
        StableMaps.dingMingCi.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.
split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }
        if(null!= cInputStringEnglish)
                {
        StableMaps.dingMingCi.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputString.sp
lit(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                }}
        studyPos.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO], cInputString
                        .split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        posCnToCn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO], cInputString
                        .split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        linkedHashMap= loopLoadForest(cInputString);
        if(null!= cInputStringKorea) {
                posCnToCn.put(cInputStringKorea.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                linkedHashMap= loopLoadForest(cInputStringKorea);
        }
        if(null!= cInputStringJapan) {
        if(!posCnToCn.containsKey(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])) {
        posCnToCn.put(cInputStringJapan.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],cInputStringJapan.split(Stabl
eData.NLP_SYMBO_SLASH).length> StableData.INT_ONE?cInputStringJapan
.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]:"未知");
                                linkedHashMap= loopLoadForest(cInputStringJapan);
                    }}
          if(null!= cInputStringTrandition) {
          if(!posCnToCn.containsKey( cInputStringTrandition.split( StableData.NLP_SYMBO_SLASH)[ StableData.INT_ZERO])) {
                    posCnToCn.put(cInputStringTrandition.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                                linkedHashMap= loopLoadForest(cInputStringTrandition);
                    }}
          if(null!= cInputStringEnglish) {
          posCnToCn.put(cInputStringEnglish.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase(),
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }}
          cReader.close();iterator<String> iterator= posCnToCn.keySet().iterator();
          while(iterator.hasNext()) {
          String key=iterator.next();if(StableData.INT_ONE== key.length()) {
          StableMaps.CiOne.put(key, posCnToCn.get(key));
}
if(StableData.INT_TWO== key.length()) {
          StableMaps.CiTwo.put(key, posCnToCn.get(key));
}
if(StableData.INT_THREE==key.length()) {
          StableMaps.CiThree.put(key, posCnToCn.get(key));
}
if(StableData.INT_FOUR== key.length()) {
          StableMaps.CiFour.put(key, posCnToCn.get(key));
}}
                                //trim StableMaps.baDongCi.remove("");
                                StableMaps.daiCi.remove("");
                                StableMaps.dingMingCi.remove("");
                                StableMaps.dongCi.remove("");
                                StableMaps.fuCi.remove("");
                                StableMaps.jieCi.remove("");
                                StableMaps.lianCi.remove("");
                                StableMaps.liangCi.remove("");
                                StableMaps.mingCi.remove("");
                                StableMaps.qingTaiCi.remove("");
                                StableMaps.shengLueCi.remove("");
                                StableMaps.shiTaiCi.remove("");
                                StableMaps.weiCi.remove("");
                                StableMaps.xianDingCi.remove("");
                                StableMaps.xingRongCi.remove("");
                                StableMaps.xingWeiCi.remove("");
                                StableMaps.zhuCi.remove("");
                                System.out.println(StableMaps.CiOne.size());
                                System.out.println(StableMaps.CiTwo.size());
                                System.out.println(StableMaps.CiThree.size());
                                System.out.println(StableMaps.CiFour.size());
                    }
public void index() throws IOException {
                                posCnToCn= new ConcurrentHashMap<>();
                                linkedHashMap= new ConcurrentHashMap<>();
                    ListCn= new CopyOnWriteArrayList<>();inputStream inputStream=
getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_CN);
                                BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream,
StableData.UTF8_STRING));
                                String cInputString;
                                Here:
while ((cInputString= cReader.readLine()) != null) {
          listCn.add(cInputString);
          if(!(!cInputString.replace(StableData.SPACE_STRING, StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                    continue Here;
          }
          if(!StableMaps.fuCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])&&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_FU)) {
          StableMaps.fuCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.dongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DONG)) {
          StableMaps.dongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.liangCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_LIANG)) {
          StableMaps.liangCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.lianCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_LIAN)) {
          StableMaps.lianCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.baDongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO])  &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_BA_DONG)) {
          StableMaps.baDongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.xianDingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&

          cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XIAN_DING))
{               StableMaps.xianDingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.mingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_MING)) {
          StableMaps.mingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
if(!StableMaps.daiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DAI)) {
          StableMaps.daiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.jieCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_JIE)) {
          StableMaps.jieCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.xingRongCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&

          cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XING_RONG))
{               StableMaps.xingRongCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.zhuCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_ZHU)) {
          StableMaps.zhuCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.weiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_WEI)) {
          StableMaps.weiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.shengLueCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&

          cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_SHENG_LUE))
{               StableMaps.shengLueCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                    cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
          }
          if(!StableMaps.qingTaiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_QING_TAI))
```

```
        {                       StableMaps.qingTaiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
        if(!StableMaps.xingWeiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_XING_WEI)) {
        StableMaps.xingWeiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
        if(!StableMaps.shiTaiCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_SHI_TAI)) {
        StableMaps.shiTaiCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
        if(!StableMaps.dingMingCi.containsKey(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]) &&

        cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].contains(StableData.NLP_CI_DING_MING))
{               StableMaps.dingMingCi.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
        posCnToCn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO], cInputString
                        .split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        linkedHashMap= loopLoadForest(cInputString);
        cReader.close();
}
public void indexFullEnToCn() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listee= listEn.iterator();
fullEnToCn= new HashMap<>();
while(listcc.hasNext()&&listee.hasNext()) {
        fullEnToCn.put(listee.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase()
                        , listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]);
}}
public void indexFullCnToEn() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listee= listEn.iterator();
fullCnToEn= new HashMap<>();
while(listcc.hasNext()&&listee.hasNext())
{
        fullCnToEn.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listee.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToFn() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listfn= listFn.iterator();
fullCnToFn= new HashMap<>();
while(listcc.hasNext()&&listfn.hasNext()) {
                        fullCnToFn.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listfn.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToKo() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listko= listKo.iterator();
fullCnToKo= new HashMap<>();
while(listcc.hasNext()&&listko.hasNext()) {
                        fullCnToKo.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listko.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToJp() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listjp= listJp.iterator();
fullCnToJp= new HashMap<>();
while(listcc.hasNext()&&listjp.hasNext()) {
                        fullCnToJp.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listjp.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToGm() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listgm= listEn.iterator();
fullCnToGm= new HashMap<>();
while(listcc.hasNext()&&listgm.hasNext()) {
                        fullCnToGm.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listgm.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
```

```
public void indexFullCnToSp() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listsp= listSp.iterator();
fullCnToSp= new HashMap<>();
while(listcc.hasNext()&&listsp.hasNext()) {
        fullCnToSp.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listsp.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToRs() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listrs= listRs.iterator();
fullCnToRs= new HashMap<>();
while(listcc.hasNext()&&listrs.hasNext()) {
                        fullCnToRs.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listrs.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToAb() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listab= listAb.iterator();
fullCnToAb= new HashMap<>();
while(listcc.hasNext()&&listab.hasNext()) {
                        fullCnToAb.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listab.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public void indexFullCnToPy() throws IOException {
Iterator<String> listcc= listCn.iterator();iterator<String> listpy= listPy.iterator();
fullCnToPy= new HashMap<>();
while(listcc.hasNext()&&listpy.hasNext()) {
                        fullCnToPy.put(listcc.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO]
                        , listpy.next().split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase());
}}
public Map<String, String> getFullEnToCn() {
return this.fullEnToCn;
                        }
public Map<String, String> getFullCnToEn() {
return this.fullCnToEn;
                        }
public void indexPosEnToCn() throws IOException {
                posEnToCn= new HashMap<>();inputStream in=
getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_EN_TO_CN);
BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
String cInputString;
Here:
        while (null!= (cInputString= cReader.readLine())) {
        if(!(!cInputString.replace(StableData.SPACE_STRING, StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
        posEnToCn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase(),
cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
cReader.close();
                        }
public void indexFn() throws IOException {
        listFn= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_FN);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listFn.add(cInputString);
        }
        cReader.close();
}
public void indexKo() throws IOException {
        listKo= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_KO);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listKo.add(cInputString);
        }
        cReader.close();
}
public void indexJp() throws IOException {
        listJp= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_JP);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listJp.add(cInputString);
        }
        cReader.close();
}
public void indexGm() throws IOException {
        ListGm= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_GM);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listGm.add(cInputString);
        }
        cReader.close();
}
public void indexSp() throws IOException {
        listSp= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_SP);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listSp.add(cInputString);
        }
        cReader.close();
}
public void indexAb() throws IOException {
        listAb= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_AB);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listAb.add(cInputString);
        }
        cReader.close();
}
public void indexRs() throws IOException {
        listRs= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_RS);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listRs.add(cInputString);
        }
        cReader.close();
}
public void indexPy() throws IOException {
        listPy= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_PY);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        while (null!= (cInputString= cReader.readLine())) {
                listPy.add(cInputString);
        }
        cReader.close();
}
public void indexPosEnToEn() throws IOException {
        posEnToEn= new HashMap<>();
        ListEn= new CopyOnWriteArrayList<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_EN_TO_EN);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        Here:
        while (null!= (cInputString= cReader.readLine())) {
                ListEn.add(cInputString);
                If(!(!cInputString.replace(StableData.SPACE_STRING,
        StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
                posEnToEn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase(),
                cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].toLowerCase());
        }
        cReader.close();
}
public void indexEnToCn() throws IOException {
        enToCn= new HashMap<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_EN_TO_CN);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        Here:
        while (null!= (cInputString= cReader.readLine())) {
                if(!(!cInputString.replace(StableData.SPACE_STRING,
        StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
        enToCn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO].toLowerCase(),
        cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
        }
        cReader.close();
}
public void indexCnToEn() throws IOException {
        cnToEn= new HashMap<>();inputStream in=
        getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_CN_TO_EN);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
        String cInputString;
        Here:
        while (null!= (cInputString= cReader.readLine())) {
                if(!(!cInputString.replace(StableData.SPACE_STRING,
        StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
        cnToEn.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
        cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE].toLowerCase());
        }
        cReader.close();
}
public Map<Long, FMHMMNode> loopLoadForest(String cInputString) {
        Here:
        for (int i= StableData.INT_ZERO;i < cInputString.length();i++) {
                if (linkedHashMap.containsKey(Long.valueOf(cInputString.charAt(i)))) {
                        FMHMMNode fHHMMNode= linkedHashMap.get(Long.valueOf(cInputString.charAt(i)));
                        linkedHashMap= doNeroPostCognitive(fHHMMNode, cInputString, i);
                        continue Here;
                }
                FMHMMNode fHHMMNode= new FMHMMNode();
                fHHMMNode.setVb(StableData.EMPTY_STRING+ cInputString.charAt(i));
                if (i+ StableData.INT_ONE < cInputString.length()) {
                        Map<String, Integer> next= new HashMap<>();
                        next.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE),
                        StableData.INT_ONE);
                        fHHMMNode.setNext(next);
                }
                linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fHHMMNode);
        }
```

```
		return linkedHashMap;
}
public Map<Long, FMHMMNode> doNeroPostCognitive(FMHMMNode fFHMMNode, String cInputString, int i) {
		if (null!= fFHMMNode.getNext()) {
				if (i+ StableData.INT_ONE < cInputString.length()) {
						linkedHashMap= doCheckAndRunNeroPostFix(fFHMMNode, cInputString, i);
				}
		return linkedHashMap;
		}
		HashMap<String, Integer> HashMap= new HashMap<>();
		if (i+ StableData.INT_ONE < cInputString.length()) {
				HashMap.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE),
				StableData.INT_ONE);
		}
		fFHMMNode.setNext(HashMap);
		linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fFHMMNode);
		return linkedHashMap;
}
public Map<Long, FMHMMNode> doCheckAndRunNeroPostFix(FMHMMNode fFHMMNode, String cInputString, int i) {
		if (!fFHMMNode.getNext().containsKey(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE))) {
				Map<String, Integer> map= fFHMMNode.getNext();
				map.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE), StableData.INT_ONE);
				fFHMMNode.setNext(map);
				linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fFHMMNode);
		}
		return linkedHashMap;
}
public Map<String, String> getPosCnToCn() {
		return this.posCnToCn;
}
public Map<String, String> getEnToCn() {
		return enToCn;
}
public Map<String, String> getCnToEn() {
		return cnToEn;
}
public Map<String, String> getPosEnToCn() {
		return this.posEnToCn;
}
public Map<String, String> getPosEnToEn() {
		return this.posEnToEn;
}
						//
if((string.charAt(i)> StableData.INT_SIXTY_FOUR && string.charAt(i)<= StableData.INT_NINTY)
		//
		|| (string.charAt(i)>= StableData.INT_NINTY_SEVEN && string.charAt(i)<= StableData.INT_ONE_TWO_TWO)
		//
		|| symbol.contains(StableData.EMPTY_STRING+ string.charAt(i)) {
				//sb.append(string.charAt(i));
//}else {
				//list.add(sb.toString().toLowerCase());
//sb.delete(StableData.INT_ZERO, sb.length());
//list.add(String.valueOf(string.charAt(i)));
//}
public List<String> englishStringToWordsList(String string) {
		List<String> list= new LinkedList<>();
		string= string.replaceAll(StableData.NLP_SPASE_REP, StableData.SPACE_STRING);
		StringBuilder sb= new StringBuilder();
		for(int i= 0;i< string.length();i++) {
		if(StopSymbol.symbol.contains(StableData.EMPTY_STRING+ string.charAt(i))) {
				List.add(sb.toString().toLowerCase());
						sb.delete(StableData.INT_ZERO, sb.length());
				List.add(String.valueOf(string.charAt(i)));
				}else {
						sb.append(string.charAt(i));
				}}
		if(StableData.INT_ZERO< sb.length()) {
```

```
		List.add(sb.toString().toLowerCase());
		}
		return list;
}
public Map<String, String> getFullCnToJp() {
		return this.fullCnToJp;
}
public Map<String, String> getFullCnToRs() {
		return this.fullCnToRs;
}
public Map<String, String> getFullCnToAb() {
		return this.fullCnToAb;
}
public Map<String, String> getFullCnToFn() {
		return this.fullCnToFn;
}
public Map<String, String> getFullCnToGm() {
		return this.fullCnToGm;
}
public Map<String, String> getFullCnToKo() {
		return this.fullCnToKo;
}
public Map<String, String> getFullCnToSp() {
		return this.fullCnToSp;
}
public Map<String, String> getFullCnToPy() {
		return this.fullCnToPy;
}
public void indexFullNegative() throws IOException {
		fullNegative= new HashMap<>();
		InputStream in= getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_NEGATIVE);
		BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
		String cInputString;
		while (null!= (cInputString= cReader.readLine())) {
		if(!fullNegative.containsKey(cInputString)) {
					fullNegative.put(cInputString, StableData.EMPTY_STRING);
			}}
		cReader.close();
}
public void indexFullPositive() throws IOException {
		fullPositive= new HashMap<>();
		InputStream in= getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_POSITIVE);
		BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StableData.UTF8_STRING));
		String cInputString;
		while (null!= (cInputString= cReader.readLine())) {
		if(!fullPositive.containsKey(cInputString)) {
					fullPositive.put(cInputString, StableData.EMPTY_STRING);
			}}
		cReader.close();
}
public Map<String, String> getFullNegative() {
		return this.fullNegative;
}
public Map<String, String> getFullPositive() {
		return this.fullPositive;
}
@Override
public Map<Long, Map<String, String>> getWordsForests() {
		Map<Long, Map<String, String>> output= new HashMap<>();
		Iterator<String> WordTree= posCnToCn.keySet().iterator();
		while(WordTree.hasNext()){
			String treeName= WordTree.next();
			if(0<treeName.length()) {
				Map<String, String> treeLeafs;
				if(output.containsKey(Long.valueOf(treeName.charAt(StableData.INT_ZERO)))){
					treeLeafs= output.get(Long.valueOf(treeName.charAt(StableData.INT_ZERO)));
					treeLeafs.put(treeName, posCnToCn.get(treeName));
```

```
                                output.put(Long.valueOf(treeName.charAt(StableData.INT_ZERO)), treeLeafs);
                        }else {
                                treeLeafs= new HashMap<>();
                        }
                        treeLeafs.put(treeName, posCnToCn.get(treeName));
                        output.put(Long.valueOf(treeName.charAt(StableData.INT_ZERO)), treeLeafs);
                }
        }
        return output;
}
public void studyNewPos(String string, String posStudy) {
        posCnToCn.put(string, posStudy);
}
@Override
public Map<String, String> getStudyPos() {
        return this.studyPos;
}}
----------------------------------------------------------------------------------------------------------------------------
package org.tinos.ortho.fhmm.imp;
import java.io.BufferedReader;
import  java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import org.tinos.ortho.fhmm.FMHMMList;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.stable.StableData;
import org.tinos.engine.euclid.imp.EuclidControllerImp;
public class FMHMMListImp implements FMHMMList {
private Map<String, String> words;
private Map<Long, FMHMMNode> linkedHashMap;
@SuppressWarnings(StableData.RAW_TYPES) private Map<Integer, Map> linkedHashMapRoot;
@SuppressWarnings(StableData.RAW_TYPES) public Map<Integer, Map> getRoot() {
                        return this.linkedHashMapRoot;
}
public void index() throws IOException {
        words= new ConcurrentHashMap<>();
        linkedHashMap= new ConcurrentHashMap<>();
        linkedHashMapRoot= new ConcurrentHashMap<>();
        InputStream inputStream= getClass().getResourceAsStream(StableData.WORDS_SOURSE_LINK_POS_CN_TO_CN);
        BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, StableData.UTF8_STRING));
        String cInputString;
        Here:
        while ((cInputString= cReader.readLine()) != null) {
                if(!(!cInputString.replace(StableData.SPACE_STRING,
        StableData.EMPTY_STRING).equals(StableData.EMPTY_STRING)
                && cInputString.split(StableData.NLP_SYMBO_SLASH).length > StableData.INT_ONE )) {
                        continue Here;
                }
                words.put(cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ZERO],
        cInputString.split(StableData.NLP_SYMBO_SLASH)[StableData.INT_ONE]);
                        linkedHashMap= loopLoadForest(cInputString);
        }
        cReader.close();
        linkedHashMapRoot= new EuclidControllerImp().mCogsEuclid(linkedHashMap);
}
public Map<Long, FMHMMNode> loopLoadForest(String cInputString) {
        Here:
        for (int i= StableData.INT_ZERO;i < cInputString.length();i++) {
                if (linkedHashMap.containsKey(Long.valueOf(cInputString.charAt(i)))) {
                        FMHMMNode fHHMMNode= linkedHashMap.get(Long.valueOf(cInputString.charAt(i)));
                        LinkedHashMap= doNeroPostCognitive(fHHMMNode, cInputString, i);
                                continue Here;
                } else {
                        FMHMMNode fHHMMNode= new FMHMMNode();
```

```
                              fHHMMNode.setVb(StableData.EMPTY_STRING+ cInputString.charAt(i));
                              if (i+ StableData.INT_ONE < cInputString.length()) {
                                        Map<String, Integer> next= new ConcurrentHashMap<>();
                                        next.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE),
                                        StableData.INT_ONE);
                                        fHHMMNode.setNext(next);
                              }
                              linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fHHMMNode);
                    }}
          return linkedHashMap;
}
public Map<Long, FMHMMNode> doNeroPostCognitive(FMHMMNode fFHMMNode, String cInputString, int i) {
          if (fFHMMNode.getNext() != null) {
          if (i+ StableData.INT_ONE < cInputString.length()) {
                    linkedHashMap= doCheckAndRunNeroPostFix(fFHMMNode, cInputString, i);
          }} else {
                    ConcurrentHashMap<String, Integer> concurrentHashMap= new ConcurrentHashMap<>();
                    if (i+ StableData.INT_ONE < cInputString.length()) {
                              concurrentHashMap.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE),
                              StableData.INT_ONE);
                    }
                    fFHMMNode.setNext(concurrentHashMap);
                    linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fFHMMNode);
          }
          return linkedHashMap;
}
public Map<Long, FMHMMNode> doCheckAndRunNeroPostFix(FMHMMNode fFHMMNode, String cInputString, int i) {
          if (!fFHMMNode.getNext().containsKey(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE))) {
                    Map<String, Integer> map= fFHMMNode.getNext();
                    Map.put(StableData.EMPTY_STRING+ cInputString.charAt(i+ StableData.INT_ONE), StableData.INT_ONE);
                    fFHMMNode.setNext(map);
                    linkedHashMap.put(Long.valueOf(cInputString.charAt(i)), fFHMMNode);
          }
          return linkedHashMap;
}
public Map<String, String> getWords() {
          return this.words;
}
public Map<Long, FMHMMNode> getMap() {
          return this.linkedHashMap;
}
public Map<String, String> getPosEnToEn() {
          return null;
}
public Map<String, String> getEnToCn() {
          return null;
}
public Map<String, String> getCnToEn() {
          return null;
}
public void indexEnToCn() throws IOException {
                    }
public void indexCnToEn() throws IOException {
                    }
public Map<String, String> getPosEnToCn() {
          return null;
}
public Map<String, String> getPosCnToCn() {
          return null;
}
public void indexPosEnToCn() throws IOException {}
public void indexPosEnToEn() throws IOException {}
public void indexPosCnToEn() throws IOException {}
public Map<String, String> getPosCnToEn() {
          return null;
}
public void indexFullEnToCn() throws IOException {}
public void indexFullCnToEn() throws IOException {}
public Map<String, String> getFullEnToCn() {
            return null;
}
public Map<String, String> getFullCnToEn() {
            return null;
}
@Override
public List<String> englishStringToWordsList(String string) {
            return null;
}
@Override
public void indexFullCnToJp() throws IOException {}
@Override
public void indexFullCnToRs() throws IOException {}
@Override
public void indexFullCnToAb() throws IOException {}
@Override
public void indexFullCnToFn() throws IOException {}
@Override
public void indexFullCnToGm() throws IOException {}
@Override
public void indexFullCnToKo() throws IOException {}
@Override
public void indexFullCnToSp() throws IOException {}
@Override
public void indexFullCnToPy() throws IOException {}
@Override
public Map<String, String> getFullCnToJp() {
            return null;
}
@Override
public Map<String, String> getFullCnToRs() {
            return null;
}
@Override
public Map<String, String> getFullCnToAb() {
            return null;
}
@Override
public Map<String, String> getFullCnToFn() {
            return null;
}
@Override
public Map<String, String> getFullCnToGm() {
            return null;
}
@Override
public Map<String, String> getFullCnToKo() {
            return null;
}
@Override
public Map<String, String> getFullCnToSp() {
            return null;
}
@Override
public Map<String, String> getFullCnToPy() {
            return null;
}
@Override
public void indexFullNegative() throws IOException {}
@Override
public void indexFullPositive() throws IOException {}
@Override
public Map<String, String> getFullNegative() {
            return null;
}
```

```
@Override
public Map<String, String> getFullPositive() {
        return null;
}
@Override
public Map<Long, FMHMMNode>[] getMaps() {
        return null;
}
@Override
public Map<Long, Map<String, String>> getWordsForests() {
        return null;
}
@Override
public void indexMixed() throws IOException {
                        // TODO Auto-generated method stub
}
@Override
public void studyNewPos(String string, String posStudy) {
                        // TODO Auto-generated method stub
}@Override
public Map<String, String> getStudyPos() {
        return null;
}
--------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.pos;
import java.util.List;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
public interface POSController {
        int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord);
        int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord);
        int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord);
        int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord);
        int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] prefixWord);
        int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord, int charPosition, String inputString);
        void addFixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords);
        int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings
                        , StringBuilder[] fixWord);
        int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
                        , int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength);
        void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord
                        , Map<String, String> wordsForest);
        int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                        , StringBuilder[] fixWord);
        int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                        , StringBuilder[] fixWord);
        int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest);
        int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
```

```
                        , String[] strings, StringBuilder[] fixWord);
        int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
                        , int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[]
nestCountInputStringLength);
        int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord);
        void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord
                        , Map<String, String> wordsForest);
        int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest);
}
---------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.pos.imp;
import java.util.List;
import java.util.Map;
import org.tinos.engine.pos.POSController;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
import org.tinos.view.stable.StableMaps;
public class POSControllerImp implements POSController{
public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord){
        if (StableMaps.xingWeiCi.containsKey(fixWord[StableData.INT_ZERO].toString())||
StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                if(StableMaps.dongCi.containsKey(strings[StableData.INT_TWO])){
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
                        fixWord);
                        return countInputStringLength;
                }
                if(StableMaps.dongCi.containsKey(strings[StableData.INT_THREE])){
                        if(StableMaps.fuCi.containsKey(StableData.EMPTY_STRING+
                        fixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))){
                                if(StableMaps.dongCi.containsKey(strings[StableData.INT_ONE])||
                                StableMaps.qingTaiCi.containsKey(strings[StableData.INT_ONE])) {
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList,
strings, fixWord);
                                        return countInputStringLength;
                                }
                                outputList.add(strings[StableData.INT_ZERO]);
                                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                                        outputList.add(strings[StableData.INT_TWO]);
                                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
                                fixWord[StableData.INT_ZERO].length());
                                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                                        return countInputStringLength;
                                }
                                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
                                fixWord[StableData.INT_ZERO].length());
                                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                                return countInputStringLength= StableData.INT_ONE;
                        }
                        if(!StableMaps.dingMingCi.containsKey(strings[StableData.INT_ZERO])){
                                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
                                        outputList, strings, fixWord);
                                        return countInputStringLength;
                                }
                                outputList.add(strings[StableData.INT_ZERO]);
                                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
                        fixWord[StableData.INT_ZERO].length());
                                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                                return countInputStringLength= StableData.INT_ONE;
                        }
                }
                if(StableMaps.mingCi.containsKey(strings[StableData.INT_TWO])){
                        if(StableData.INT_ZERO< fixWord[StableData.INT_ONE].length()&&
                                StableMaps.zhuCi.containsKey(StableData.EMPTY_STRING
                                        + fixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))){
                                if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                        if(!StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                                                countInputStringLength= parserFirstCharOfThree(countInputStringLength,
                                                outputList, strings, fixWord);
                                                return countInputStringLength;
                                        }
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
                                        outputList, strings, fixWord);
                                        return countInputStringLength;
                                }
                                outputList.add(strings[StableData.INT_ZERO]);
                                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
                                fixWord[StableData.INT_ZERO].length());
                                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                                return countInputStringLength= StableData.INT_ONE;
                        }
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
                fixWord);
                        return countInputStringLength;
                }
                if(StableMaps.mingCi.containsKey(strings[StableData.INT_ONE])||
                StableMaps.fuCi.containsKey(strings[StableData.INT_ONE])){
                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings,
                fixWord);
                        return countInputStringLength;
                }
                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings,
                fixWord);
                        return countInputStringLength;
                }
                outputList.add(strings[StableData.INT_ZERO]);
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                countInputStringLength= StableData.INT_ONE;
                        return countInputStringLength;
                }
                if(StableMaps.dongCi.containsKey(strings[StableData.INT_THREE])){
                        if(StableMaps.dongCi.containsKey(strings[StableData.INT_TWO])){
                                if(StableMaps.mingCi.containsKey(strings[StableData.INT_ZERO])) {
                                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList,
                                        strings,   fixWord);
                                        return countInputStringLength;
                                }
                        }
                        if(null!= fixWord[StableData.INT_ZERO]&&
                                StableData.INT_ZERO<fixWord[StableData.INT_ZERO].length()){
                                        if(StableMaps.zhuCi.containsKey(StableData.EMPTY_STRING+
                                fixWord[StableData.INT_ZERO].charAt(StableData.INT_ZERO))){
                                                if(!StableMaps.mingCi.containsKey(strings[StableData.INT_ONE])) {
                                                        countInputStringLength=
        parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                                        return countInputStringLength;
                                                }
                                        }
                        }
                        if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])) {
                                countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings,
        fixWord);
```

```
                                        return countInputStringLength;
                              }
                              countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                              return countInputStringLength;
                    }
                    if(StableMaps.mingCi.containsKey(strings[StableData.INT_TWO])){
                                        if(StableData.INT_ZERO< fixWord[StableData.INT_ONE].length()&&
                                                  StableMaps.zhuCi.containsKey(StableData.EMPTY_STRING
                                        + fixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))){
                                                  if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                                  if(!StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                                                            countInputStringLength= parserFirstCharOfThree(countInputStringLength,
                                                  outputList, strings, fixWord);
                                                            return countInputStringLength;
                                                  }
                                                  countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
                              outputList, strings, fixWord);
                                                  return countInputStringLength;
                                        }
                                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList,
                                        strings,   fixWord);
                                        return countInputStringLength;
                    }
                    if(StableData.INT_ZERO< fixWord[StableData.INT_ONE].length()&&
                              StableMaps.dingMingCi.containsKey(StableData.EMPTY_STRING
                                                  + fixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))){
                              countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength,
                              outputList, strings, fixWord);
                              return countInputStringLength;
                    }
                    if(StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                              if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                                            countInputStringLength=
                    parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                                        return countInputStringLength;
                              }
                    }
                    if(StableData.INT_ZERO== fixWord[StableData.INT_ZERO].length()
                                        || StableMaps.jieCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                              if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings,
fixWord);
                                        return countInputStringLength;
                              }
                    }
                    countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                    return countInputStringLength;
          }
          if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                    outputList.add(strings[StableData.INT_ONE]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
          fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                    countInputStringLength= StableData.INT_TWO;
                    return countInputStringLength;
          }
          outputList.add(strings[StableData.INT_ZERO]);
          if(StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])) {
                    outputList.add(strings[StableData.INT_TWO]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                    return countInputStringLength;
          }
          fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
          fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
          return countInputStringLength= StableData.INT_ONE;
}
```

```
public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings, 
StringBuilder[] prefixWord) {
		if ((StableMaps.mingCi.containsKey(strings[StableData.INT_TWO].toString())) && 
(StableMaps.jieCi.containsKey(prefixWord[StableData.INT_ZERO].toString())
				|| StableMaps.xingWeiCi.containsKey(prefixWord[StableData.INT_ZERO].toString())
				|| StableMaps.dongCi.containsKey(prefixWord[StableData.INT_ZERO].toString())))){
					countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
					return countInputLength;
				}
		if (StableMaps.dongCi.containsKey(strings[StableData.INT_TWO].toString())
				||StableMaps.liangCi.containsKey(strings[StableData.INT_TWO].toString())) {
					countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
					return countInputLength;
				}
		if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
					outputList.add(strings[StableData.INT_ONE]);
					prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, 
prefixWord[StableData.INT_ZERO].length());
					prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
					countInputLength= StableData.INT_TWO;
					return countInputLength;
				}
				outputList.add(strings[StableData.INT_ZERO]);
		if(StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])) {
					outputList.add(strings[StableData.INT_TWO]);
					prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, 
prefixWord[StableData.INT_ZERO].length());
					prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
					return countInputLength;
				}
				prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
				prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
				return countInputLength= StableData.INT_ONE;
		}
public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,String[] strings, 
StringBuilder[] prefixWord) {
		if (StableMaps.fuCi.containsKey(strings[StableData.INT_TWO].toString())) {
				if (StableMaps.fuCi.containsKey(prefixWord[StableData.INT_ZERO].toString())) {
				countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
				return countInputLength;
					}}
		if (StableMaps.dongCi.containsKey(strings[StableData.INT_TWO].toString())) {
				if (StableMaps.zhuCi.containsKey(StableData.EMPTY_STRING+ 
prefixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))
				|| StableMaps.daiCi.containsKey(StableData.EMPTY_STRING+ 
prefixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))) {

				countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
				return countInputLength;
					}}
		if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])) {
					countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);
					return countInputLength;
				}
				outputList.add(strings[StableData.INT_ZERO]);
				prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
				prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
				return StableData.INT_ONE;
		}
public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings, 
StringBuilder[] prefixWord){
		if (!wordsForest.containsKey(prefixWord[StableData.INT_ZERO].toString())){
					return countInputStringLength;
				}
		if (StableMaps.daiCi.containsKey(prefixWord[StableData.INT_ZERO].toString())
				||StableMaps.fuCi.containsKey(prefixWord[StableData.INT_ZERO].toString())) {
					countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);
```

```
                                return countInputStringLength;
                        }
        if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                outputList.add(strings[StableData.INT_ONE]);
                                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                                prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                                return countInputStringLength;
                        }
                        return countInputStringLength- StableData.INT_TWO;
        }
public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                                , String[] strings, StringBuilder[] fixWord, int charPosition, String inputString){
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                        if (StableMaps.liangCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                        countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                                }
                        if (StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                        ||StableMaps.xingRongCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                        ||StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                        ||StableMaps.zhuCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                        ||StableMaps.liangCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
        if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])) {
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                        outputList.add(strings[StableData.INT_ONE]);
                        return countInputStringLength;
                        }
                        countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                                }
                                addFixWordsOfTwo(charPosition, inputString, fixWord);if (StableData.INT_ZERO<
fixWord[StableData.INT_ONE].length()&& StableMaps.fuCi.containsKey(StableData.EMPTY_STRING
+ fixWord[StableData.INT_ONE].toString().charAt(StableData.INT_ZERO))){
                        countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                                }
                        if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                        outputList.add(strings[StableData.INT_ONE]);
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                        return countInputStringLength;
                                }
                                countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);
                                return countInputStringLength;
                        }
                        return countInputStringLength;
        }
public void addFixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {
                        fixWords[StableData.INT_ONE].delete(StableData.INT_ZERO, fixWords[StableData.INT_ONE].length());
        if (charPosition+ StableData.INT_SEVEN < inputString.length()) {
                                fixWords[StableData.INT_ONE].append(inputString.substring(charPosition+ StableData.INT_TWO
                        , charPosition+ StableData.INT_SEVEN));
                                return;
                        }
                        fixWords[StableData.INT_ONE].append(inputString.substring(charPosition+ StableData.INT_TWO
                        , inputString.length()));
        }
public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings
                                , StringBuilder[] fixWord){
                        outputList.add(strings[StableData.INT_ZERO]);
                        String postNext=StableData.EMPTY_STRING+ strings[StableData.INT_ONE].charAt(StableData.INT_ONE);
                        outputList.add(postNext);
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(postNext);
                        return countInputStringLength;
        }
public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord) {
        if (StableMaps.lianCi.containsKey(strings[StableData.INT_TWO])){
                        countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
                }
        if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                        countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
                }
                return countInputLength;
        }
public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
        if (outputList.size() == StableData.INT_ZERO){
                        didNotFindFirstChar(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString()) &&
(StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())||
StableMaps.fuCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.daiCi.containsKey(fixWord[StableData.INT_ZERO].toString()) ||
StableMaps.weiCi.containsKey(fixWord[StableData.INT_ZERO].toString()))){
                        countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                }
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString()) &&
(StableMaps.zhuCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.shengLueCi.containsKey(fixWord[StableData.INT_ZERO].toString()))){
                        for (int BackPosition= StableData.INT_ZERO;
                BackPosition < fixWord[StableData.INT_ONE].length();
                BackPosition++){

                int[] nestCountInputStringLength= new int[StableData.INT_ONE];
                int result= loopCheckBackFix(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings
                        , nestCountInputStringLength);
        if (result == StableData.INT_RIGHT){
                return nestCountInputStringLength[StableData.INT_ZERO];
                }}
                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                outputList.add(strings[StableData.INT_ONE]);
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                return countInputStringLength- StableData.INT_ONE;
                        }
                        return countInputStringLength- StableData.INT_THREE;
                }
        if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                        outputList.add(strings[StableData.INT_ONE]);
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        return countInputStringLength- StableData.INT_ONE;
                }
                return countInputStringLength- StableData.INT_THREE;
        }
public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
                        , int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength){
                String charPositionAtFixWord= StableData.EMPTY_STRING+ fixWord[StableData.INT_ONE].charAt(backPosition);
        if (wordsForest.containsKey(charPositionAtFixWord) && (StableMaps.zhuCi.containsKey(charPositionAtFixWord)
                || StableMaps.shengLueCi.containsKey(charPositionAtFixWord)||
StableMaps.fuCi.containsKey(charPositionAtFixWord))){
                if(!wordsForest.get(fixWord[StableData.INT_ZERO].toString()).contains(StableData.NLP_CI_SHENG_LUE) &&
wordsForest.get(charPositionAtFixWord).contains(StableData.NLP_CI_FU)){

                return StableData.INT_ERROR;
                        }
```

```
                                nestCountInputStringLength[StableData.INT_ZERO]= parserFirstCharOfThree(countInputStringLength,
outputList
                    , strings, fixWord);
                                return StableData.INT_RIGHT;
                    }
                    return StableData.INT_ERROR;
          }
public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord
                                , Map<String, String> wordsForest){
          if(!StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                    if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                    outputList.add(strings[StableData.INT_ONE]);
                    outputList.add(strings[StableData.INT_THREE]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_THREE]);
                                }
                                return;
                    }
          if (StableMaps.fuCi.containsKey(strings[StableData.INT_TWO])){
                                outputList.add(strings[StableData.INT_ZERO]);
                                outputList.add(strings[StableData.INT_TWO]);
                                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                                return;
                    }
if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                                outputList.add(strings[StableData.INT_ONE]);
                                outputList.add(strings[StableData.INT_THREE]);
                                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_THREE]);
                    }}
public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                                , StringBuilder[] fixWord){
                    outputList.add(strings[StableData.INT_ZERO]);
                    outputList.add(strings[StableData.INT_TWO]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                    return countInputStringLength;
          }
public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings
                                , StringBuilder[] fixWord){
                    outputList.add(strings[StableData.INT_ONE]);
                    outputList.add(strings[StableData.INT_THREE]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_THREE]);
                    return countInputStringLength;
          }
public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                                , String[] strings, StringBuilder[] fixWord){
          if (StableData.INT_ZERO== outputList.size()){
                                didNotFindFirstChar(outputList, strings, fixWord, wordsForest);
                                return countInputStringLength;
                    }
          if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                    if (StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
          if(StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])) {
                    countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                    return countInputStringLength;
                    }
                    outputList.add(strings[StableData.INT_ZERO]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                    return countInputStringLength= StableData.INT_ONE;
                                } else if(fixWord[StableData.INT_ONE].length()> StableData.INT_ONE) {
                    String firstChar= StableData.EMPTY_STRING+
fixWord[StableData.INT_ONE].toString().charAt(StableData.INT_ZERO);
                    String secondChar= StableData.EMPTY_STRING+
fixWord[StableData.INT_ONE].toString().charAt(StableData.INT_ONE);
        if(!StableMaps.fuCi.containsKey(firstChar)&& !StableMaps.fuCi.containsKey(secondChar)&&!StableMaps.fuCi.containsKey(first
Char
+ secondChar)) {
        if(StableMaps.CiOne.containsKey(firstChar)&& StableMaps.CiOne.containsKey(secondChar)) {
                        outputList.add(strings[StableData.INT_ZERO]);
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                        countInputStringLength=
StableData.INT_ONE;if(StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])) {
outputList.add(strings[StableData.INT_TWO]);
fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
countInputStringLength= StableData.INT_THREE;
                        }
                        return countInputStringLength;
                }}}
                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
                outputList.add(strings[StableData.INT_ONE]);
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                return countInputStringLength- StableData.INT_ONE;
                        }
                if (StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                outputList.add(strings[StableData.INT_ZERO]);
                outputList.add(strings[StableData.INT_TWO]);
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                return countInputStringLength;
                        }}
                return countInputStringLength;
        }
public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
        if (StableData.INT_ZERO== outputList.size()&&
(StableMaps.CiTwo.get(strings[StableData.INT_TWO]).contains(StableData.NLP_CI_WEI))){
                        outputList.add(strings[StableData.INT_ZERO]);
                        outputList.add(strings[StableData.INT_TWO]);
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                        return countInputStringLength;
                }
        if (outputList.size() > StableData.INT_ZERO&& wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                if (StableMaps.qingTaiCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.weiCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.lianCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
                countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                return countInputStringLength;
                        } else{
                        if(StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())) {
        if(StableMaps.xingWeiCi.containsKey(strings[StableData.INT_ONE])
|| StableMaps.xingRongCi.containsKey(strings[StableData.INT_ONE])) {
                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                }}
        if (StableMaps.mingCi.containsKey(strings[StableData.INT_TWO])){
                outputList.add(strings[StableData.INT_ZERO]);
                outputList.add(strings[StableData.INT_TWO]);
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                return countInputStringLength;
                }else if (StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])){
        if(StableMaps.jieCi.containsKey(strings[StableData.INT_ONE])) {
                        countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                        return countInputStringLength;
                }
                outputList.add(strings[StableData.INT_ONE]);
```

```
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                    countInputStringLength= StableData.INT_TWO;
                    return countInputStringLength;
                    }else if (StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                    outputList.add(strings[StableData.INT_ZERO]);
                    outputList.add(strings[StableData.INT_TWO]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                    return countInputStringLength;
                    }}}
          if(StableMaps.jieCi.containsKey(strings[StableData.INT_ONE])) {
                              countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);
                              return countInputStringLength;
                    }
                    outputList.add(strings[StableData.INT_ZERO]);
          if(StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])) {
                              outputList.add(strings[StableData.INT_TWO]);
                              fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                              fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                              return countInputStringLength;
                    }
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                    return countInputStringLength= StableData.INT_ONE;
          }
public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength
                              , String[] strings, StringBuilder[] fixWord){
          if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                    if (StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())||
StableMaps.daiCi.containsKey(fixWord[StableData.INT_ZERO].toString())){

                    countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);
                    return countInputStringLength;
                              }
                    if (StableMaps.liangCi.containsKey(strings[StableData.INT_ONE])){
                    outputList.add(strings[StableData.INT_ONE]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                    return StableData.INT_TWO;
                              }
                    if
((StableMaps.xingWeiCi.containsKey(fixWord[StableData.INT_ZERO].toString())||StableMaps.xingRongCi.containsKey(fixWord[Stab leD
                               ata.INT_ZERO].toString()))
                    && StableMaps.mingCi.containsKey(strings[StableData.INT_ONE])){
                    outputList.add(strings[StableData.INT_ONE]);
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                    return StableData.INT_TWO;
                              }}
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                    outputList.add(strings[StableData.INT_ZERO]);
          if (StableMaps.CiTwo.containsKey(strings[StableData.INT_TWO])){
                              outputList.add(strings[StableData.INT_TWO]);
                              fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                              fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                              return StableData.INT_THREE;
                    }
                    return StableData.INT_ONE;
          }
          //
public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                              , String[] strings, StringBuilder[] fixWord){
          if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                    if (StableMaps.liangCi.containsKey(fixWord[StableData.INT_ZERO].toString())){
```

```
                countInputStringLength= parserFirstCharOfTwoForMap(countInputStringLength, outputList, strings, fixWord
                        , wordsForest);
                return countInputStringLength;
                        }countInputStringLength -= StableData.INT_TWO;if (wordsForest.containsKey(strings[StableData.INT_ONE])){
        if (outputList.containsKey(strings[StableData.INT_ONE])){
                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                } else{
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_ONE]);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                countInputStringLength += StableData.INT_TWO;
                        }
                        return countInputStringLength;
                }
                return countInputStringLength;
        }
public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest){
                countInputStringLength -= StableData.INT_TWO;
        if (outputList.containsKey(strings[StableData.INT_ZERO])){
                        WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ZERO]);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                } else{
                        WordFrequency wordFrequency= new WordFrequency();
                        wordFrequency.setFrequency(StableData.INT_ONE);
                        wordFrequency.setWord(strings[StableData.INT_ZERO]);
                        outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                countInputStringLength += StableData.INT_ONE;
                return countInputStringLength;
        }
public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                if (StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())||
StableMaps.daiCi.containsKey(fixWord[StableData.INT_ZERO].toString())){

                countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord
                        , wordsForest);
                return countInputStringLength;
                        } countInputStringLength -= StableData.INT_THREE;if
(wordsForest.containsKey(strings[StableData.INT_ONE])){
        if (outputList.containsKey(strings[StableData.INT_ONE])){
                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                } else{
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_ONE]);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                countInputStringLength += StableData.INT_TWO;
                        }
```

```
                    return countInputStringLength;
               }
               return countInputStringLength;
       }
public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                    , String[] strings, StringBuilder[] fixWord){
       if (outputList.size()== StableData.INT_ZERO&& StableMaps.weiCi.containsKey(strings[StableData.INT_TWO])){
               if (outputList.containsKey(strings[StableData.INT_ZERO])){

               WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ZERO]);
               wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
               outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                    } else{

               WordFrequency wordFrequency= new WordFrequency();
               wordFrequency.setFrequency(StableData.INT_ONE);
               wordFrequency.setWord(strings[StableData.INT_ZERO]);
               outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                    }
               if (outputList.containsKey(strings[StableData.INT_TWO])){

               WordFrequency wordFrequency= outputList.get(strings[StableData.INT_TWO]);
               wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
               outputList.put(strings[StableData.INT_TWO], wordFrequency);
                    } else{

               WordFrequency wordFrequency= new WordFrequency();
               wordFrequency.setFrequency(StableData.INT_ONE);
               wordFrequency.setWord(strings[StableData.INT_TWO]);
               outputList.put(strings[StableData.INT_TWO], wordFrequency);
                    }
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                    return countInputStringLength;
               }
       if (outputList.size()> StableData.INT_ZERO&& wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
               if (StableMaps.lianCi.containsKey(fixWord[StableData.INT_ZERO].toString())||
StableMaps.qingTaiCi.containsKey(fixWord[StableData.INT_ZERO].toString())
               || StableMaps.weiCi.containsKey(fixWord[StableData.INT_ZERO].toString())){

               countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord,
wordsForest);
               return countInputStringLength;
                    } else{

               countInputStringLength-= StableData.INT_THREE;
       if (wordsForest.containsKey(strings[StableData.INT_ONE])){
       if (outputList.containsKey(strings[StableData.INT_ONE])){
                    WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                    wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                    outputList.put(strings[StableData.INT_ONE], wordFrequency);
               } else{
                    WordFrequency wordFrequency= new WordFrequency();
                    wordFrequency.setFrequency(StableData.INT_ONE);
                    wordFrequency.setWord(strings[StableData.INT_ONE]);
                    outputList.put(strings[StableData.INT_ONE], wordFrequency);
               }
               fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
               fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
               countInputStringLength+= StableData.INT_TWO;
               }
               return countInputStringLength;
                    }}
               return countInputStringLength;
       }
public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                    , String[] strings, StringBuilder[] fixWord){
        if (outputList.size()== StableData.INT_ZERO){
                    didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);
                    return countInputStringLength;
                }
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())
                && (StableMaps.mingCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                    || StableMaps.daiCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                    || StableMaps.weiCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                    || StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                    || StableMaps.fuCi.containsKey(fixWord[StableData.INT_ZERO].toString()))){
                    countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord,
wordsForest);
                    return countInputStringLength;
                }
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString()) &&
(StableMaps.zhuCi.containsKey(fixWord[StableData.INT_ZERO].toString())
                || StableMaps.shengLueCi.containsKey(fixWord[StableData.INT_ZERO].toString()))){
                    for (int BackPosition= StableData.INT_ZERO;
                BackPosition< fixWord[StableData.INT_ONE].length();
                BackPosition++){
                int[] nestCountInputStringLength= new int[StableData.INT_ONE];
                int result= loopCheckBackFixForMap(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings
                    , nestCountInputStringLength);
        if (result== StableData.INT_RIGHT){
                return nestCountInputStringLength[StableData.INT_ZERO];
                }}
                    countInputStringLength-= StableData.INT_THREE;if
(wordsForest.containsKey(strings[StableData.INT_ONE])){
        if (outputList.containsKey(strings[StableData.INT_ONE])){
                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                } else{
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_ONE]);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                countInputStringLength += StableData.INT_TWO;
                    }
                    return countInputStringLength;
                }
                countInputStringLength-= StableData.INT_THREE;
        if (wordsForest.containsKey(strings[StableData.INT_ONE])){
                if (outputList.containsKey(strings[StableData.INT_ONE])){

                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                    } else{

                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_ONE]);
                outputList.put(strings[StableData.INT_ONE], wordFrequency);
                    }
                    fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                    countInputStringLength+= StableData.INT_TWO;
                }
                return countInputStringLength;
        }
public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest
                    , int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings, int[]
```

```
nestCountInputStringLength){
                String charPositionAtFixWord= StableData.EMPTY_STRING+ fixWord[StableData.INT_ONE].charAt(backPosition);
        if (wordsForest.containsKey(charPositionAtFixWord)&& (StableMaps.zhuCi.containsKey(charPositionAtFixWord)
                || wordsForest.get(charPositionAtFixWord).contains(StableData.NLP_CI_SHENG_LUE))){
                        nestCountInputStringLength[StableData.INT_ZERO]= parserFirstCharOfThreeForMap(countInputStringLength,
outputList
                , strings, fixWord, wordsForest);
                        return StableData.INT_RIGHT;
                }
                return StableData.INT_ERROR;
        }
public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength
                        , String[] strings, StringBuilder[] fixWord){
        if (StableData.INT_ZERO== outputList.size()){
                        didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);
                        return countInputStringLength;
                }
        if (wordsForest.containsKey(fixWord[StableData.INT_ZERO].toString())){
                if (StableMaps.dongCi.containsKey(fixWord[StableData.INT_ZERO].toString())){

                countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord,
wordsForest);
                return countInputStringLength;
                        } else{

                countInputStringLength-= StableData.INT_THREE;
        if (wordsForest.containsKey(strings[StableData.INT_ONE])){
        if (outputList.containsKey(strings[StableData.INT_ONE])){
                        WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ONE]);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(strings[StableData.INT_ONE], wordFrequency);
                } else{
                        WordFrequency wordFrequency= new WordFrequency();
                        wordFrequency.setFrequency(StableData.INT_ONE);
                        wordFrequency.setWord(strings[StableData.INT_ONE]);
                        outputList.put(strings[StableData.INT_ONE], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ONE]);
                countInputStringLength+= StableData.INT_TWO;
                }
                return countInputStringLength;
                        }}
                return countInputStringLength;
        }
public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord
                        , Map<String, String> wordsForest){
        if(!wordsForest.containsKey(strings[StableData.INT_TWO])){
                        return;
                }
        if (StableMaps.fuCi.containsKey(strings[StableData.INT_TWO])){
                if (outputList.containsKey(strings[StableData.INT_ZERO])){

                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ZERO]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                        } else{

                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_ZERO]);
                outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                        }
                if (outputList.containsKey(strings[StableData.INT_TWO])){

                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_TWO]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_TWO], wordFrequency);
                        } else{

                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_TWO]);
                outputList.put(strings[StableData.INT_TWO], wordFrequency);
                        }
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                }}
public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings
                        , StringBuilder[] fixWord, Map<String, String> wordsForest){
                countInputStringLength-= StableData.INT_THREE;
        if (outputList.containsKey(strings[StableData.INT_ZERO])){
                        WordFrequency wordFrequency= outputList.get(strings[StableData.INT_ZERO]);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                } else{
                        WordFrequency wordFrequency= new WordFrequency();
                        wordFrequency.setFrequency(StableData.INT_ONE);
                        wordFrequency.setWord(strings[StableData.INT_ZERO]);
                        outputList.put(strings[StableData.INT_ZERO], wordFrequency);
                }
                fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                fixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                countInputStringLength ++;
        if (wordsForest.containsKey(strings[StableData.INT_TWO])){
                if (outputList.containsKey(strings[StableData.INT_TWO])){

                WordFrequency wordFrequency= outputList.get(strings[StableData.INT_TWO]);
                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                outputList.put(strings[StableData.INT_TWO], wordFrequency);
                        } else{

                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(strings[StableData.INT_TWO]);
                outputList.put(strings[StableData.INT_TWO], wordFrequency);
                        }
                        fixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWord[StableData.INT_ZERO].length());
                        fixWord[StableData.INT_ZERO].append(strings[StableData.INT_TWO]);
                        countInputStringLength+= StableData.INT_TWO;
                        return countInputStringLength;
                }
                return countInputStringLength;
        }}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.pos.imp;
import java.util.List;
import java.util.Map;
import org.tinos.engine.pos.POSController;
import org.tinos.view.obj.WordFrequency;
public class POSControllerCognitionImp implements POSController{
        @Override
public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] prefixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord, int charPosition, String inputString) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public void addFixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {
```

```
                // TODO Auto-generated method stub
        }
        @Override
public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength,
List<String> outputList, String[] strings, int[] nestCountInputStringLength) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {
                // TODO Auto-generated method stub
        }
        @Override
public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,
StringBuilder[] fixWord, Map<String, String> wordsForest) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int
countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,
                        int[] nestCountInputStringLength) {
                // TODO Auto-generated method stubreturn 0;
        }
        @Override
public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int
countInputStringLength, String[] strings, StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings, StringBuilder[] fixWord, Map<String,
String> wordsForest) {
                // TODO Auto-generated method stub
        }
        @Override
public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,
StringBuilder[] fixWord, Map<String, String> wordsForest) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings,
StringBuilder[] fixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord) {
                // TODO Auto-generated method stub return 0;
        }
        @Override
public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength, String[] strings,
StringBuilder[] prefixWord) {
                // TODO Auto-generated method stubreturn 0;
        }}
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nlp;
import java.util.List;
import java.util.Map;
import org.tinos.engine.pos.POSController;
import org.tinos.view.obj.WordFrequency;
public interface NLPController {
public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , POSController posUtils, int charPosition, String textInputString);
public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList
                        , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                        , POSController posUtils, int charPosition, String textInputString);
public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder, Map<String, String> wordsForest,
StringBuilder[] prefixWord, POSController posUtils, int charPosition, String textInputString);
public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder
                        , Map<String, String> wordsForest, StringBuilder[] prefixWord, POSController posUtils, int charPosition, String
textInputString);
public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , POSController posUtils);
public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList
                        , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                        , POSController posUtils);
public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder
                                , Map<String, String> wordsForest, StringBuilder[] prefixWord, POSController posUtils);
}
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nlp.imp;
import java.util.List;
import java.util.Map;
import org.tinos.engin.utils.WordForestUtil;
import org.tinos.engine.nlp.NLPController;
import org.tinos.engine.pos.POSController;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
import org.tinos.view.stable.StableMaps;
public class NLPControllerImp implements NLPController{
public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList
                                , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                                , POSController posUtils, int charPosition, String textInputString){
                        String countWordNode= stringBuilder.toString();
            if (StableData.INT_ZERO== prefixWord[StableData.INT_ZERO].length()){
                        if(StableMaps.CiTwo.containsKey(countWordNode)) {
                                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                                prefixWord[StableData.INT_ZERO].append(countWordNode);
                                outputList.add(countWordNode);
                                return countInputStringLength;
                        }
                        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                        prefixWord[StableData.INT_ZERO].append(stringBuilder.charAt(StableData.INT_ZERO));
                        outputList.add(StableData.EMPTY_STRING+ stringBuilder.charAt(StableData.INT_ZERO));
                        return countInputStringLength- StableData.INT_ONE;
            }
            String[] strings= new String[StableData.INT_TWO];
            strings[StableData.INT_ZERO]= String.valueOf(countWordNode.charAt(StableData.INT_ZERO));
            strings[StableData.INT_ONE]= countWordNode;
            if (StableMaps.mingCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList, countInputStringLength
                        , strings, prefixWord, charPosition, textInputString);
                                return countInputStringLength;
                        }
            if (StableMaps.baDongCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputStringLength= posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList, countInputStringLength
                        , strings, prefixWord);
                                return countInputStringLength;
                        }
            if (StableMaps.jieCi.containsKey(strings[StableData.INT_ZERO].toString())){
                        if (StableMaps.dongCi.containsKey(prefixWord[StableData.INT_ZERO].toString())){
            if (!StableMaps.jieCi.containsKey(countWordNode)){
                        countInputStringLength= posUtils.parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);
                        return countInputStringLength;
                        }}}
            if (StableMaps.CiTwo.containsKey(countWordNode)){
                                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                                prefixWord[StableData.INT_ZERO].append(countWordNode);
                                outputList.add(countWordNode);
                                return countInputStringLength;
                        }
                        countInputStringLength= posUtils.parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);
                        return countInputStringLength;
            }public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList
                                , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                                , POSController posUtils, int charPosition, String textInputString){
                        String inputString= stringBuilder.toString();
            if (StableMaps.CiThree.containsKey(inputString)){
                                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                                prefixWord[StableData.INT_ZERO].append(inputString);
                                outputList.add(inputString);
                                return countInputLength;
                        }
                        String[] strings= new String[StableData.INT_FOUR];
                        strings[StableData.INT_ZERO]= String.valueOf(inputString.charAt(StableData.INT_ZERO));
                        strings[StableData.INT_ONE]= String.valueOf(inputString.charAt(StableData.INT_ZERO))
            + inputString.charAt(StableData.INT_ONE);
                        strings[StableData.INT_TWO]= String.valueOf(inputString.charAt(StableData.INT_ONE))
            + inputString.charAt(StableData.INT_TWO);
                        strings[StableData.INT_THREE]= String.valueOf(inputString.charAt(StableData.INT_TWO));
            if (null== prefixWord[StableData.INT_ZERO]){
                        if (StableMaps.CiThree.containsKey(inputString)){
                        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                        prefixWord[StableData.INT_ZERO].append(inputString);
                        outputList.add(inputString);
                        return countInputLength;
                                }
                                StringBuilder stringsBuilder= new StringBuilder();
                                countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                        , stringsBuilder.append(strings[StableData.INT_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);
                                return countInputLength;
                        }
            if (!StableMaps.CiOne.containsKey(strings[StableData.INT_ZERO])){
                                StringBuilder stringsBuilder= new StringBuilder();
                                countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList
                        , stringsBuilder.append(strings[StableData.INT_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);
                                return countInputLength;
                        }
            if(StableMaps.lianCi.containsKey(strings[StableData.INT_THREE])) {
                                countInputLength= posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if (StableMaps.lianCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if (StableMaps.jieCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);
                                return countInputLength;
                        }if (StableMaps.zhuCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if (StableMaps.liangCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if (StableMaps.mingCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if (StableMaps.shiTaiCi.containsKey(strings[StableData.INT_ZERO])){
                                countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength, strings,
        prefixWord);
                                return countInputLength;
                        }
            if
        (StableMaps.dongCi.containsKey(strings[StableData.INT_ZERO])||StableMaps.fuCi.containsKey(strings[StableData.INT_ZERO])){
                        if(StableMaps.zhuCi.containsKey(prefixWord[StableData.INT_ZERO].toString())
                        && (StableMaps.liangCi.containsKey(strings[StableData.INT_TWO])
        || StableMaps.jieCi.containsKey(strings[StableData.INT_TWO]))) {
                        countInputLength= posUtils.parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
                        return countInputLength;
```

```
                    }
        if(StableMaps.fuCi.containsKey(strings[StableData.INT_TWO])||StableMaps.mingCi.containsKey(strings[StableData.INT_TWO])
                ||StableMaps.daiCi.containsKey(strings[StableData.INT_TWO])) {

                countInputLength= posUtils.parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);
                return countInputLength;
                        }}
        if (StableMaps.fuCi.containsKey(strings[StableData.INT_ZERO])){
                        countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);
                        return countInputLength;
                }
        if(StableMaps.CiTwo.containsKey(strings[StableData.INT_ONE])) {
                        StringBuilder stringsBuilder= new StringBuilder();
                        countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList,
stringsBuilder.append(strings[StableData.INT_ONE])
                , wordsForest, prefixWord, posUtils, charPosition, textInputString);
                        return countInputLength;
                }
                outputList.add(strings[StableData.INT_ZERO]);
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(strings[StableData.INT_ZERO]);
                return StableData.INT_ONE;
        }
        // 2 个月研究发现 词性越来越多, 根据笛摩根定律, 先把未知词汇也添加到条件中. 之后采用 排除法优化.
        //
        if(StableMaps.jieCi.containsKey(preRegister)|| StableMaps.mingCi.containsKey(preRegister)||
StableMaps.xingRongCi.containsKey(preRegister)
                        //
                        || StableMaps.fuCi.containsKey(preRegister)|| StableMaps.dongCi.containsKey(preRegister)||
StableMaps.lianCi.containsKey(preRegister)
                        //
                        || StableMaps.liangCi.containsKey(preRegister)|| StableMaps.xingWeiCi.containsKey(preRegister)||
StableMaps.shiTaiCi.containsKey(preRegister)
                        //
                        || StableMaps.zhuCi.containsKey(preRegister)) {
                //if(StableMaps.mingCi.containsKey(postRegister)|| StableMaps.dongCi.containsKey(postRegister)||
StableMaps.lianCi.containsKey(postRegister)
                //
                || StableMaps.xingRongCi.containsKey(postRegister)|| StableMaps.xingWeiCi.containsKey(postRegister)||
StableMaps.liangCi.containsKey(preRegister)
                //
                || StableMaps.fuCi.containsKey(postRegister)|| StableMaps.jieCi.containsKey(postRegister)) {
public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder, Map<String, String> wordsForest,
StringBuilder[] prefixWord, POSController posUtils, int charPosition, String textInputString){
                        String inputString= stringBuilder.toString();
        if (StableMaps.CiFour.containsKey(inputString)){
                output.add(inputString);
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(inputString);
                return countInputStringLength;
        }//will make pre 3 or post 3 check. now finished pre 3 .20190330
        String preRegister= StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO)+
inputString.charAt(StableData.INT_ONE);
        String inRegister= StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ONE)+
inputString.charAt(StableData.INT_TWO);
        String postRegister= StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_TWO)+
inputString.charAt(StableData.INT_THREE);
        if(StableMaps.dongCi.containsKey(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_THREE)
+ prefixWord[StableData.INT_ONE].charAt(StableData.INT_ZERO))) {
                countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils, charPosition,
textInputString);
                return countInputStringLength;
        }
        if (StableMaps.CiTwo.containsKey(preRegister)){
        if (StableMaps.CiTwo.containsKey(postRegister)){
                        String string= StableData.EMPTY_STRING+
inputString.charAt(StableData.INT_ZERO);if(StableMaps.xingWeiCi.containsKey(prefixWord[StableData.INT_ZERO].toString())
&&StableMaps.shiTaiCi.containsKey(string)) {
                output.add(string);
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(string);
                return countInputStringLength- StableData.INT_THREE;
                        }
                if(StableMaps.zhuCi.containsKey(string)){
                String[] strings= new String[StableData.INT_FOUR];
                strings[StableData.INT_ZERO]= String.valueOf(inputString.charAt(StableData.INT_ZERO));
                strings[StableData.INT_ONE]= String.valueOf(inputString.charAt(StableData.INT_ZERO))+
inputString.charAt(StableData.INT_ONE);
                strings[StableData.INT_TWO]= String.valueOf(inputString.charAt(StableData.INT_ONE))
+ inputString.charAt(StableData.INT_TWO);
                strings[StableData.INT_THREE]= String.valueOf(inputString.charAt(StableData.INT_TWO));
                countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output, countInputStringLength-
StableData.INT_ONE, strings, prefixWord);
                return countInputStringLength;
                        }
                        output.add(preRegister);
                        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                        prefixWord[StableData.INT_ZERO].append(preRegister);
                        return countInputStringLength-StableData.INT_TWO;
                }}
        if(StableMaps.CiThree.containsKey(preRegister+
                        inputString.charAt(StableData.INT_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(preRegister+ inputString.charAt(StableData.INT_TWO));
                output.add(preRegister+ inputString.charAt(StableData.INT_TWO));
                return countInputStringLength- StableData.INT_ONE ;
        }
        if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {
                countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils, charPosition,
textInputString);
                return countInputStringLength;
        }
        if(StableMaps.CiTwo.containsKey(preRegister)) {
                countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength- StableData.INT_TWO, output
                , stringBuilder.delete(StableData.INT_TWO, StableData.INT_FOUR), wordsForest, prefixWord, posUtils, charPosition,
textInputString);
                return countInputStringLength;
        }
        output.add(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
        prefixWord[StableData.INT_ZERO].append(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
        return countInputStringLength= StableData.INT_ONE;
}
//卡诺图化简.PCA 阀门分流. 卷积催化, .原来备注这里 ,20190523
//
if(!wordsForest.containsKey(preRegister)&& (wordsForest.containsKey(inRegister)||wordsForest.containsKey(postRegister))) {
                        //
        if(wordsForest.containsKey(preRegister+ inputString.charAt(StableData.INT_TWO))) {
                //
                output.add(preRegister+ inputString.charAt(StableData.INT_TWO));
                //
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                //
                prefixWord[StableData.INT_ZERO].append(preRegister+ inputString.charAt(StableData.INT_TWO));
                //
                return countInputStringLength- StableData.INT_ONE;
                //
        }
        //
        output.add(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
        //
        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
        prefixWord[StableData.INT_ZERO].append(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO));
        // return countInputStringLength- StableData.INT_THREE;
        //}
        //if(wordsForest.containsKey(preRegister)&& wordsForest.containsKey(inRegister+ inputString.charAt(StableData.INT_THREE)))
{
                        //
        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                        //
                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils, charPosition,
textInputString);
        //
        return countInputStringLength;
        //}
        //if(wordsForest.containsKey(preRegister)) {
                        //
        countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                        //
                        , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils, charPosition,
textInputString);
        //
        return countInputStringLength;
        //}
public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder
                        , Map<String, String> wordsForest, StringBuilder[] prefixWord, POSController posUtils, int charPosition, String
textInputString){
                String inputString= stringBuilder.toString();
        if (wordsForest.containsKey(inputString)){
                        output.add(inputString);
                        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                        prefixWord[StableData.INT_ZERO].append(inputString);
                        return countInputStringLength;
                }
                countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest
                , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils, charPosition,
textInputString);
                return countInputStringLength;
        }
public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList
                        , StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord
                        , POSController posUtils){
                String countWordNode= stringBuilder.toString();
        if (!wordsForest.containsKey(countWordNode)){
                        WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);
                        return --countInputStringLength;
                }
        if (prefixWord[StableData.INT_ZERO].length()== StableData.INT_ZERO){
                        WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);
                        return countInputStringLength;
                }
                String[] strings= new String[StableData.INT_TWO];
                strings[StableData.INT_ZERO]= String.valueOf(countWordNode.charAt(StableData.INT_ZERO));
                strings[StableData.INT_ONE]= String.valueOf(countWordNode.charAt(StableData.INT_ZERO))
        + String.valueOf(countWordNode.charAt(StableData.INT_ONE));
        if (wordsForest.containsKey(strings[StableData.INT_ZERO])){
                if (wordsForest.get(strings[StableData.INT_ZERO]).contains(StableData.NLP_CI_MING)){
                countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList, countInputStringLength
                        , strings, prefixWord);
                return countInputStringLength;
                        }}
        if (wordsForest.containsKey(strings[StableData.INT_ONE])){
                        WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);
                        return countInputStringLength;
                }
                return StableData.INT_ZERO;
        }
```

```
public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList
                                , Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord
                                , POSController posUtils){
                    String inputString= stringBuilder.toString();
          if (wordsForest.containsKey(inputString)){
                              WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
                              return countInputLength;
                    }
                    String[] strings= new String[StableData.INT_FOUR];
                    strings[StableData.INT_ZERO]= String.valueOf(inputString.charAt(StableData.INT_ZERO));
                    strings[StableData.INT_ONE]= String.valueOf(inputString.charAt(StableData.INT_ZERO))
          + inputString.charAt(StableData.INT_ONE);
                    strings[StableData.INT_TWO]= String.valueOf(inputString.charAt(StableData.INT_ONE)
          + inputString.charAt(StableData.INT_TWO));
                    strings[StableData.INT_THREE]= String.valueOf(inputString.charAt(StableData.INT_TWO));
          if (null== prefixWord[StableData.INT_ZERO]){
                    if (wordsForest.containsKey(inputString)){
                    WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);
                    return countInputLength;
                              }
                              StringBuilder stringsBuilder= new StringBuilder();
                              countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                    , stringsBuilder.append(strings[StableData.INT_ONE]), wordsForest, prefixWord, posUtils);
                              return countInputLength;
                    }
          if (!wordsForest.containsKey(strings[StableData.INT_ZERO])){
                              StringBuilder stringsBuilder= new StringBuilder();
                              countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                    , stringsBuilder.append(strings[StableData.INT_ONE]), wordsForest, prefixWord, posUtils);
                              return countInputLength;
                    }
          if (StableMaps.zhuCi.containsKey(strings[StableData.INT_ZERO])){
                              countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength
                    , strings, prefixWord);
                              return countInputLength;
                    }
          if (StableMaps.liangCi.containsKey(strings[StableData.INT_ZERO])){
                              countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength
                    , strings, prefixWord);
                              return countInputLength;
                    }
          if (StableMaps.zhuCi.containsKey(strings[StableData.INT_ZERO])){
                              countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength
                    , strings, prefixWord);
                              return countInputLength;
                    }
          if (StableMaps.lianCi.containsKey(strings[StableData.INT_ZERO])){
                              countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength
                    , strings, prefixWord);
                              return countInputLength;
                    }
                    StringBuilder stringsBuilder= new StringBuilder();
                    countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList
                    , stringsBuilder.append(strings[StableData.INT_ONE]), wordsForest, prefixWord, posUtils);
                    return countInputLength;
          }
public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder
                                , Map<String, String> wordsForest, StringBuilder[] prefixWord, POSController posUtils){
                    String inputString= stringBuilder.toString();
          if (wordsForest.containsKey(inputString)){
                              WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);
                              return countInputStringLength;
                    }
          if(StableMaps.mingCi.containsKey(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_ZERO)+
inputString.charAt(StableData.INT_ONE))) {
                    if(StableMaps.mingCi.containsKey(StableData.EMPTY_STRING+ inputString.charAt(StableData.INT_TWO)+
inputString.charAt(StableData.INT_THREE))) {
```

```
                    WordForestUtil.wordsForestContainsKey(output, StableData.EMPTY_STRING+
inputString.charAt(StableData.INT_ZERO)+ inputString.charAt(StableData.INT_ONE), prefixWord);
                    return countInputStringLength;
                              }}
                    countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output, wordsForest
                    , stringBuilder.delete(StableData.INT_THREE, StableData.INT_FOUR), prefixWord, posUtils);
                    return countInputStringLength;
          }}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nero;
import java.util.Map;
import org.tinos.view.obj.FMHMMNode;
public interface NEROControllerOneTime {
                              StringBuilder getBinaryForestsRecurWordOneTime(StringBuilder inputStringWordNode, String inputString, int
charPosition
                              , int inputStringLength, Map<Long, FMHMMNode>[] forestsRoots, int forestDepth, int charPositionNext);
          StringBuilder getBinaryForestRecurWordOneTime(StringBuilder inputStringWordNode, String inputString, int charPosition, int
inputStringLength, Map<Long, FMHMMNode> forestRoots, int forestDepth, int charPositionNext);
          StringBuilder getQuickForestRecurWord(StringBuilder inputStringWordNode, String inputString, int charPosition
                              , int inputStringLength, Map<String, String> forestRoots, int forestDepth, int charPositionNext);
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nero;
import java.util.Map;
import org.tinos.view.stable.StableData;
public interface NEROController {
                              @SuppressWarnings(StableData.RAW_TYPES)
          StringBuilder getBinaryForestRecurWord(StringBuilder inputStringWordNode, String inputString, int charPosition
                              , int inputStringLength, Map<Integer, Map> forestRoots, int forestDepth, int charPositionNext);
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nero.imp;
import java.util.Map;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.stable.StableData;
import org.tinos.engine.nero.NEROControllerOneTime;
public class NEROControllerOneTimeImp implements NEROControllerOneTime {
public StringBuilder getBinaryForestRecurWordOneTime(StringBuilder outputWordNode, String inputString
                              , int charPosition, int inputStringLength, Map<Long, FMHMMNode> forestRoots, int forestDepth
                              , int charPositionNext) {
          if (StableData.INT_THREE== forestDepth){
                              return outputWordNode;
                    }
                    FMHMMNode fFHMMNode= forestRoots.get(Long.valueOf(inputString.charAt(charPosition)));
          if (null== fFHMMNode) {
                              return outputWordNode;
                    }
          Map<String, Integer> outputList= fFHMMNode.getNext();
          if (null== outputList || charPositionNext>= inputStringLength) {
                              return outputWordNode;
                    }
                    char positionOfi= inputString.charAt(charPositionNext);
          if (outputList.containsKey(String.valueOf(positionOfi))) {
                              outputWordNode= getBinaryForestRecurWordOneTime(outputWordNode.append(positionOfi), inputString,
charPositionNext
                    , inputStringLength, forestRoots, ++forestDepth, ++charPositionNext);
                    }
                    return outputWordNode;
          }
          //prepare for the big map collection in the future.
public StringBuilder getBinaryForestsRecurWordOneTime(StringBuilder outputWordNode, String inputString
                              , int charPosition, int inputStringLength, Map<Long, FMHMMNode>[] forestsRoots, int forestDepth
                              ,int charPositionNext) {
          if (StableData.INT_THREE== forestDepth){
                              return outputWordNode;
                    }FMHMMNode fFHMMNode= getFMHMMNode(forestsRoots,inputString,charPosition);
```

```
        if (null== fFHMMNode) {
                        return outputWordNode;
                }
        Map<String, Integer> outputList= fFHMMNode.getNext();
        if (null== outputList|| charPositionNext>= inputStringLength) {
                        return outputWordNode;
                }
                char positionOfi= inputString.charAt(charPositionNext);
        if (outputList.containsKey(String.valueOf(positionOfi))) {
                        outputWordNode= getBinaryForestsRecurWordOneTime(outputWordNode.append(positionOfi), inputString,
charPositionNext
                , inputStringLength, forestsRoots, ++forestDepth, ++charPositionNext);
                }
                return outputWordNode;
        }
        private FMHMMNode getFMHMMNode(Map<Long, FMHMMNode>[] forestsRoots, String inputString, int charPosition) {
                for(Map<Long, FMHMMNode> forestsRoot: forestsRoots) {
                if(forestsRoot.containsKey(Long.valueOf(inputString.charAt(charPosition)))){

                return forestsRoot.get(Long.valueOf(inputString.charAt(charPosition)));
                        }}
                return null;
        }
public StringBuilder getQuickForestRecurWord(StringBuilder outputWordNode, String inputString, int charPosition
                        , int inputStringLength, Map<String, String> posCntoCn, int forestDepth, int charPositionNext ) {
        if (StableData.INT_THREE== forestDepth|| charPositionNext>= inputStringLength) {
                        return outputWordNode;
                }
                char positionOfi= inputString.charAt(charPositionNext);
        if (posCntoCn.containsKey(String.valueOf(outputWordNode.toString()+ positionOfi))) {
                        outputWordNode= getQuickForestRecurWord(outputWordNode.append(positionOfi), inputString
                , charPositionNext, inputStringLength, posCntoCn, ++forestDepth, ++charPositionNext);
                }
                return outputWordNode;
        }}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.nero.imp;
import java.util.Map;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.stable.StableData;
import org.tinos.engine.nero.NEROController;
public class NEROControllerImp implements NEROController {
        @SuppressWarnings({
                StableData.RAW_TYPES, StableData.UNCHECKED})
public StringBuilder getBinaryForestRecurWord(StringBuilder outputWordNode, String inputString, int charPosition
                        , int inputStringLength, Map<Integer, Map> forestRoots, int forestDepth, int charPositionNext ) {
        if (StableData.INT_THREE== forestDepth) {
                        return outputWordNode;
                }
                char charAtPosition= inputString.charAt(charPosition);
                int rangeHigh= charAtPosition>> StableData.INT_TEN;
                Map<Integer, Map> trees= forestRoots.get(rangeHigh);if (null== trees) {

                return outputWordNode;
                        }
                        int range= charAtPosition>> StableData.INT_SIX;if (!trees.containsKey(range)) {

                return outputWordNode;
                        }
                Map<Long, FMHMMNode> maps= trees.get(range);
                        FMHMMNode fFHMMNode= maps.get(Long.valueOf(charAtPosition));if (null== fFHMMNode) {

                return outputWordNode;
                        }
                Map<String, Integer> outputList= fFHMMNode.getNext();if (null== outputList||charPositionNext>= inputStringLength) {
                return outputWordNode;
                        }
```

```
                                char positionOfi= inputString.charAt(charPositionNext);if (outputList.containsKey(String.valueOf(positionOfi)))
{

                    outputWordNode= getBinaryForestRecurWord(outputWordNode.append(positionOfi), inputString
                                , charPositionNext, inputStringLength, forestRoots
                                , forestDepth+ StableData.INT_ONE, ++charPositionNext);
                                }
                                return outputWordNode;
          }}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.liner;
import java.util.List;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
public interface Quick6DLuoYaoguangSort {
                    void quick6DLuoYaoGuangSortWordFrequency(List<WordFrequency> list, int leftPosition, int rightPosition);
          int partition(List<WordFrequency> list, int leftPosition, int rightPosition);
          List<WordFrequency> frequencyWordMapToList(Map<String, WordFrequency> map);
          void quick6DLuoYaoGuangSortWordFrequency(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition);
          int partition(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition);
          Map<Integer, WordFrequency> frequencyWordMapToMap(Map<String, WordFrequency> map);
}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.liner.imp;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.tinos.engine.liner.Quick6DLuoYaoguangSort;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
/*
 ** 快排 6 小高峰修正算法 作者 罗瑶光
 */
public class Quick6DLuoYaoguangSortImp implements Quick6DLuoYaoguangSort {
public void quick6DLuoYaoGuangSortWordFrequency(List<WordFrequency> list, int leftPosition, int rightPosition) {
          if (leftPosition < rightPosition) {
                                int c= rightPosition - leftPosition+ StableData.INT_ONE;if (c < StableData.INT_FOUR) {
                    int j;
                    for (int i= StableData.INT_ONE+ leftPosition;i < leftPosition+ c;i++) {
                    j= i;
                    while (j >= StableData.INT_ONE+ leftPosition) {
                    if (list.get(j).getFrequency() < list.get(j - StableData.INT_ONE).getFrequency()) {
WordFrequency wordFrequency= list.get(j);
list.set(j, list.get(j - StableData.INT_ONE));
list.set(j - StableData.INT_ONE, wordFrequency);
                                }
                                j--;
                    }}
                    return;
                                }
                                int pos= partition(list, leftPosition, rightPosition);
                                quick6DLuoYaoGuangSortWordFrequency(list, leftPosition, pos - StableData.INT_ONE);
                                quick6DLuoYaoGuangSortWordFrequency(list, ++ pos, rightPosition);
                    }}
public int partition(List<WordFrequency> list, int leftPosition, int rightPosition) {
                    int rightPositionNew= rightPosition;
                    int leftPositionNew= leftPosition;
                    WordFrequency wordFrequencyX= list.get(leftPosition);
                    WordFrequency wordFrequencyY= list.get(rightPosition);
                    //小高峰修正边缘均衡开始
          if (wordFrequencyX.getFrequency()<= wordFrequencyY.getFrequency()) {
                                wordFrequencyY= wordFrequencyX;
                    }
                    //小高峰修正边缘均衡结束
                    while (leftPositionNew< rightPositionNew) {
                                while ((list.get(leftPositionNew).getFrequency()<= wordFrequencyY.getFrequency()) && (leftPositionNew<
rightPositionNew)){

                    leftPositionNew++;
                              }
                              while (list.get(rightPositionNew).getFrequency()> wordFrequencyY.getFrequency()) {
                    rightPositionNew--;
                              }
                    if (leftPositionNew< rightPositionNew){

                    WordFrequency wordFrequency= list.get(rightPositionNew);
          List.set(rightPositionNew, list.get(leftPositionNew));
          List.set(leftPositionNew, wordFrequency);
                              }}
          List.set(leftPosition, list.get(rightPositionNew));
          List.set(rightPositionNew, wordFrequencyY);
                    return rightPositionNew;
          }
          @SuppressWarnings(StableData.RAW_TYPES)
public List<WordFrequency> frequencyWordMapToList(Map<String, WordFrequency> map) {
          List<WordFrequency> list= new ArrayList<>();
                    Iterator iterator= map.keySet().iterator();
                    while (iterator.hasNext()) {
                    List.add(map.get(iterator.next()));
                    }
                    return list;
          }
          @Override
public void quick6DLuoYaoGuangSortWordFrequency(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition) {
                              }
          @Override
public int partition(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition) {
                    return StableData.INT_ZERO;
          }
          @Override
public Map<Integer, WordFrequency> frequencyWordMapToMap(Map<String, WordFrequency> map) {
                    return null;
          }}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.liner.imp;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
//import java.util.concurrent.ConcurrentHashMap;
import org.tinos.engine.liner.Quick6DLuoYaoguangSort;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
public class Quick6DLuoYaoguangSort3DMapImp implements Quick6DLuoYaoguangSort {
          @Override
public void quick6DLuoYaoGuangSortWordFrequency(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition) {
          if (leftPosition< rightPosition) {
                              int c= rightPosition- leftPosition+ StableData.INT_ONE;if (c< StableData.INT_FOUR) {

                    int j;
                    for (int i= StableData.INT_ONE+ leftPosition;i< leftPosition+ c;i++) {
                    j= i;
                    while (j>= StableData.INT_ONE+ leftPosition) {
                    if (map.get(j).getFrequency()< map.get(j- StableData.INT_ONE).getFrequency()) {
WordFrequency wordFrequency= map.get(j);
map.put(j, map.get(j- StableData.INT_ONE));
map.put(j- StableData.INT_ONE, wordFrequency);
                              }
                              j--;
                    }}
                    return;
                              }
                              int pos= partition(map, leftPosition, rightPosition);
                    quick6DLuoYaoGuangSortWordFrequency(map, leftPosition, pos- StableData.INT_ONE);
                    quick6DLuoYaoGuangSortWordFrequency(map, ++pos, rightPosition);
            }}
        @Override
public int partition(Map<Integer, WordFrequency> map, int leftPosition, int rightPosition) {
            int leftPositionNew= leftPosition;
            WordFrequency wordFrequencyX= map.get(leftPosition);
            WordFrequency wordFrequencyY= map.get(rightPosition);
            wordFrequencyY= wordFrequencyX.getFrequency()<= wordFrequencyY.getFrequency()
            ? wordFrequencyX: wordFrequencyY;
            while (leftPositionNew< rightPosition) {
                    while ((map.get(leftPositionNew).getFrequency()<= wordFrequencyY.getFrequency()) && (leftPositionNew<
rightPosition)) {

            leftPositionNew++;
                    }
                    while (map.get(rightPosition).getFrequency()> wordFrequencyY.getFrequency()) {
            rightPosition--;
                    }
            if (leftPositionNew< rightPosition) {

            WordFrequency wordFrequency= map.get(rightPosition);
        Map.put(rightPosition, map.get(leftPositionNew));
        Map.put(leftPositionNew, wordFrequency);
                    }}
        Map.put(leftPosition, map.get(rightPosition));
        Map.put(rightPosition, wordFrequencyY);
            return rightPosition;
        }
        @SuppressWarnings(StableData.RAW_TYPES)
public Map<Integer, WordFrequency> frequencyWordMapToMap(Map<String, WordFrequency> map) {
        Map<Integer, WordFrequency> listMap= new HashMap<>();
            Iterator iterator= map.keySet().iterator();
            int c= StableData.INT_ZERO;
            while (iterator.hasNext()) {
            ListMap.put(c++, map.get(iterator.next()));
            }
            return listMap;
        }
public void quick6DLuoYaoGuangSortWordFrequency(List<WordFrequency> list, int leftPosition, int rightPosition) {
                    }
public int partition(List<WordFrequency> list, int leftPosition, int rightPosition) {
            return StableData.INT_ZERO;
        }
public List<WordFrequency> frequencyWordMapToList(Map<String, WordFrequency> map) {
            return null;
        }}
---------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.euclid;
import java.util.Map;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.stable.StableData;
public interface EuclidController {
                    @SuppressWarnings({
            StableData.RAW_TYPES})
        Map<Integer, Map> mCogsEuclid(Map<Long, FMHMMNode> concurrentHashMap);
}
---------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.euclid.imp;
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;
//import java.util.concurrent.HashMap;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.stable.StableData;
import org.tinos.engine.euclid.EuclidController;
public class EuclidControllerImp implements EuclidController {
	@SuppressWarnings({
		StableData.RAW_TYPES, StableData.UNCHECKED}) public Map<Integer, Map> mCogsEuclid(Map<Long, 
FMHMMNode> HashMap) {
	Map<Integer, Map> HashMapRoot= new HashMap<>();
		Iterator<Long> iter= HashMap.keySet().iterator();
		Here:
			while (iter.hasNext()) {
		Long keyValue= iter.next();
		Integer charOfKeyValueToInteger= Integer.valueOf(StableData.EMPTY_STRING+ keyValue);
		int range= (charOfKeyValueToInteger.intValue()>> StableData.INT_SIX);
		int rangeHigh= range >> StableData.INT_FOUR;
	if (!HashMapRoot.containsKey(rangeHigh)) {
			HashMap<Long, FMHMMNode> innerHashMap= new HashMap<>();innerHashMap.put(keyValue, 
HashMap.get(keyValue));
			HashMap<Integer, HashMap> root= new HashMap<>();
			root.put(range, innerHashMap);
			HashMapRoot.put(rangeHigh, root);
			continue Here;
		}
	Map<Integer, HashMap> root= HashMapRoot.get(rangeHigh);
	if (!root.containsKey(range)) {
			HashMap<Long, FMHMMNode> innerHashMap= new HashMap<>();innerHashMap.put(keyValue, 
HashMap.get(keyValue));
			root.put(range, innerHashMap);
			HashMapRoot.put(rangeHigh, root);
			continue Here;
		}
		HashMap<Long, FMHMMNode> innerHashMap= root.get(range);
		innerHashMap.put(keyValue, HashMap.get(keyValue));
		root.put(range, innerHashMap);
		HashMapRoot.put(rangeHigh, root);
			}
		return HashMapRoot;
	}}
-------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.analysis;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
public interface Analyzer {
		void init() throws IOException;
	void initMixed() throws IOException;
	List<String> parserString(String input);
	void addFixWords(int charPosition, String inputString,StringBuilder[] fixWords);
	Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException;
	List<WordFrequency> sortWordFrequencyMap(Map<String,WordFrequency> map) throws IOException;
	List<WordFrequency> getWordFrequency(List<String> sets)throws IOException;
	Map<Integer,WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException;
	Map<Integer,WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String,WordFrequency> map);
	Map<Integer,WordFrequency> sortWordFrequencyMapToSortMap(Map<String,WordFrequency> map);
	Map<String,WordFrequency> parserStringByReturnFrequencyMap(String inputString);
	Map<String,String> getPosEnToCn();
	Map<String,String> getPosEnToEn();
	Map<String,String> getPosCnToCn();
	Map<String,String> getEnToCn();
	Map<String,String> getCnToEn();
	Map<String,String> getFullEnToCn();
	Map<String,String> getFullCnToEn();
	String[] parserEnglishString(String englishString);
	List<String> parserMixedString(String mixedString);
	Map<String,WordFrequency> parserMixStringByReturnFrequencyMap(String key);
	void studyNewWord(String study, String token, String posStudy);
	Map<String, String> getStudyPos();
}
-------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.analysis.imp;
```

```
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.Iterator;
import java.util.LinkedList;
import org.tinos.view.obj.FMHMMNode;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
import org.tinos.ortho.fhmm.FHMMList;
import org.tinos.ortho.fhmm.imp.FMHMMListOneTimeImp;
import org.tinos.engine.nero.NEROControllerOneTime;
import org.tinos.engine.nero.imp.NEROControllerOneTimeImp;
import org.tinos.engine.nlp.NLPController;
import org.tinos.engine.nlp.imp.NLPControllerImp;
import org.tinos.engine.pos.POSController;
import org.tinos.engine.pos.imp.POSControllerImp;
import org.tinos.engin.utils.WordFrequencyUtil;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.liner.Quick6DLuoYaoguangSort;
import org.tinos.engine.liner.imp.Quick6DLuoYaoguangSort3DMapImp;
public class AnalyzerImp implements Analyzer {
                    protected FHMMList fHMMList;
      protected NEROControllerOneTime neroController;
      protected NLPController nlpController;
      protected POSController posController;
      protected Quick6DLuoYaoguangSort quick6DLuoYaoguangSort;
      protected Map<Long, FMHMMNode> forestRoots;
      protected Map<Long, Map<String, String>> wordsForests;
      protected Map<Long, FMHMMNode> []forestsRoots;
      protected Map<String, String> wordsForest;
public void init() throws IOException {
            this.fHMMList= new FMHMMListOneTimeImp();
            fHMMList.index();
            fHMMList.indexPosEnToCn();
            fHMMList.indexPosEnToEn();
            fHMMList.indexEnToCn();
            fHMMList.indexCnToEn();
            fHMMList.indexFullEnToCn();
            fHMMList.indexFullCnToEn();
            neroController= new NEROControllerOneTimeImp();
            nlpController= new NLPControllerImp();
            posController= new POSControllerImp();
            quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMapImp();
            forestRoots=fHMMList.getMap();
            forestsRoots=fHMMList.getMaps();
            wordsForest=fHMMList.getPosCnToCn();
            wordsForests=fHMMList.getWordsForests();
      }
public void initMixed() throws IOException {
            this.fHMMList=new FMHMMListOneTimeImp();
            fHMMList.indexMixed();
            fHMMList.indexPosEnToCn();
            fHMMList.indexPosEnToEn();
            fHMMList.indexEnToCn();
            fHMMList.indexCnToEn();
            fHMMList.indexFullEnToCn();
            fHMMList.indexFullCnToEn();
            neroController= new NEROControllerOneTimeImp();
            nlpController= new NLPControllerImp();
            posController= new POSControllerImp();
            quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMapImp();
            forestRoots=fHMMList.getMap();
            forestsRoots=fHMMList.getMaps();
            wordsForest=fHMMList.getPosCnToCn();
            wordsForests=fHMMList.getWordsForests();
      }
```

```
public List<String> parserMixedString(String mixedString) {
                    mixedString+= StableData.SPACE_STRING_DISTINCTION;
                    int inputStringLength= mixedString.length();
          List<String> outputList= new LinkedList<>();
                    int forestDepth= StableData.INT_ZERO;
                    int countInputStringLength;
                    StringBuilder[] fixWords= new StringBuilder[StableData.INT_TWO];
                    fixWords[StableData.INT_ZERO]= new StringBuilder();
                    fixWords[StableData.INT_ONE]= new StringBuilder();
                    StringBuilder stringBuilder= new StringBuilder();
                    int find= StableData.INT_ZERO;
                    Here:
                              for (int charPosition= StableData.INT_ZERO;
                    charPosition<inputStringLength;
                    charPosition
+=(countInputStringLength==StableData.INT_ZERO?StableData.INT_ONE:countInputStringLength)) {
          if(mixedString.charAt(charPosition) < StableData.INT_TEN_SOUTHANDS && charPosition < inputStringLength -
StableData.INT_ONE){
          if(find == StableData.INT_ZERO) {
                              fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    }
                    fixWords[StableData.INT_ZERO].append(mixedString.charAt(charPosition));
                    countInputStringLength= StableData.INT_ONE;
                    find= StableData.INT_ONE;
                    continue Here;
                    }
          if(StableData.INT_ONE == find) {
                    find= StableData.INT_ZERO;
                    Iterator<String> it =
fHMMList.englishStringToWordsList(fixWords[StableData.INT_ZERO].toString()).iterator();
                    StringBuilder number= new StringBuilder();
                    while(it.hasNext()) {
                              String temp= it.next();if(StableData.NUMBERS.contains(temp)) {
number.append(temp);
                              }else {
if(number.length()>0) {
          outputList.add(number.toString());
          number.delete(0, number.length());
}
outputList.add(temp);
                              }}
          if(number.length()>0) {
                              outputList.add(number.toString());
                              number.delete(0, number.length());
                    }
                    fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    }
                    stringBuilder.delete(StableData.INT_ZERO, stringBuilder.length());
                    stringBuilder= neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString
                              .charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth
                              , charPosition+ StableData.INT_ONE);
                    String countWordNode= stringBuilder.toString();
                    int compare= countInputStringLength= countWordNode.length();
          if (StableData.INT_ONE == compare) {
                    outputList.add(countWordNode);
                    fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    fixWords[StableData.INT_ZERO].append(countWordNode);
                    continue Here;
                    }
          if (StableData.INT_TWO == compare) {
                    countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList
, stringBuilder, wordsForest, fixWords, posController, charPosition, mixedString);
                    continue Here;
                    }
          if (StableData.INT_THREE == compare) {
                    addFixWords(charPosition, mixedString, fixWords);
                    countInputStringLength= nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList
, wordsForest, stringBuilder, fixWords, posController, charPosition, mixedString);
                    continue Here;
                    }
          if (StableData.INT_FOUR == compare) {
                    addFixWords(charPosition, mixedString, fixWords);
                    countInputStringLength= nlpController.doSlangCheck(countInputStringLength, outputList, stringBuilder
, wordsForest, fixWords, posController, charPosition, mixedString);
                    }}
                    return outputList;
          }
public List<String> parserString(String inputString) {
          List<String> outputList= new LinkedList<>();
                    int inputStringLength= inputString.length();
                    int forestDepth= StableData.INT_ZERO;
                    int countInputStringLength;
                    StringBuilder[] fixWords= new StringBuilder[StableData.INT_TWO];
                    fixWords[StableData.INT_ZERO]= new StringBuilder();
                    fixWords[StableData.INT_ONE]= new StringBuilder();
                    StringBuilder stringBuilder= new StringBuilder();
                    int find= StableData.INT_ZERO;
                    Here:
                              for (int charPosition= StableData.INT_ZERO;
                    charPosition< inputStringLength;
                    charPosition
+= (countInputStringLength!= StableData.INT_ZERO? countInputStringLength: StableData.INT_ONE)) {
          if(StableData.INT_ONE_TWO_EIGHT> inputString.charAt(charPosition)){
          if(fixWords[StableData.INT_ZERO].length()> StableData.INT_ZERO) {
                    if(fixWords[StableData.INT_ZERO].charAt(fixWords[StableData.INT_ZERO].length()- StableData.INT_ONE)
          < StableData.INT_ONE_TWO_EIGHT) {
fixWords[StableData.INT_ZERO].append(inputString.charAt(charPosition));
countInputStringLength= StableData.INT_ONE;
find= StableData.INT_ONE;
continue Here;
                              }
                              fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    }
                    find= StableData.INT_ONE;
                    fixWords[StableData.INT_ZERO].append(inputString.charAt(charPosition));
                    countInputStringLength= StableData.INT_ONE;
                    continue Here;
                    }
          if(find== StableData.INT_ONE) {
                    find= StableData.INT_ZERO;
                    outputList.add(fixWords[StableData.INT_ZERO].toString());
                    }
                    stringBuilder.delete(StableData.INT_ZERO, stringBuilder.length());
                    stringBuilder= neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString
                              .charAt(charPosition)), inputString, charPosition, inputStringLength, forestRoots, forestDepth
                              , charPosition+ StableData.INT_ONE);
                    String countWordNode= stringBuilder.toString();
                    int compare= countInputStringLength= countWordNode.length();
          if (compare== StableData.INT_ONE) {
                    outputList.add(countWordNode);
                    fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    fixWords[StableData.INT_ZERO].append(countWordNode);
                    continue Here;
                    }
          if (compare== StableData.INT_TWO) {
                    countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList
, stringBuilder, wordsForest, fixWords, posController, charPosition, inputString);
                    continue Here;
                    }
          if (compare== StableData.INT_THREE) {
                    addFixWords(charPosition, inputString, fixWords);
                    countInputStringLength= nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList
, wordsForest, stringBuilder, fixWords, posController, charPosition, inputString);
                    continue Here;
                    }
          if (compare== StableData.INT_FOUR) {
                    addFixWords(charPosition, inputString, fixWords);
                    countInputStringLength= nlpController.doSlangCheck(countInputStringLength, outputList, stringBuilder
, wordsForest, fixWords, posController, charPosition, inputString);
                    }}
                    return outputList;
          }
public Map<String, WordFrequency> parserStringByReturnFrequencyMap(String inputString) {
          Map<String, String> wordsForest= fHMMList.getPosCnToCn();
          Map<String, WordFrequency> outputList= new ConcurrentHashMap<>();
          Map<Long, FMHMMNode> forestRoots= fHMMList.getMap();
                    int inputStringLength= inputString.length();
                    int forestDepth= StableData.INT_ZERO;
                    int countInputStringLength;
                    StringBuilder[] fixWords= new StringBuilder[StableData.INT_TWO];
                    fixWords[StableData.INT_ZERO]= new StringBuilder();
                    fixWords[StableData.INT_ONE]= new StringBuilder();
                    StringBuilder stringBuilder= new StringBuilder();
                    int find= StableData.INT_ZERO;
                    Here:for (int charPosition= StableData.INT_ZERO;
                    charPosition< inputStringLength;
                    charPosition
          += (countInputStringLength== StableData.INT_ZERO? StableData.INT_ONE: countInputStringLength)) {
                    if(inputString.charAt(charPosition)< StableData.INT_ONE_TWO_EIGHT){
          if(fixWords[StableData.INT_ZERO].length()> StableData.INT_ZERO) {
          if(fixWords[StableData.INT_ZERO].charAt(fixWords[StableData.INT_ZERO].length()- StableData.INT_ONE)
< StableData.INT_ONE_TWO_EIGHT) {
                              fixWords[StableData.INT_ZERO].append(inputString.charAt(charPosition));
                              countInputStringLength= StableData.INT_ONE;
                              find= StableData.INT_ONE;
                              continue Here;
                    }
                    fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    }
                    find=StableData.INT_ONE;
                    fixWords[StableData.INT_ZERO].append(inputString.charAt(charPosition));
                    countInputStringLength= StableData.INT_ONE;
                    continue Here;
                              }
                    if(find == StableData.INT_ONE) {
                    find= StableData.INT_ZERO;
                    WordFrequencyUtil.WordFrequencyFindCheck(outputList, fixWords);
                              }
                              stringBuilder.delete(StableData.INT_ZERO, stringBuilder.length());
                              stringBuilder=
neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString.charAt(charPosition)), inputString, charPosition,
inputStringLength, forestRoots, forestDepth
                    , charPosition+ StableData.INT_ONE);
                              String countWordNode= stringBuilder.toString();int compare= countInputStringLength=
countWordNode.length();if (compare == StableData.INT_ONE) {

                    WordFrequencyUtil.WordFrequencyCompareCheck(outputList, fixWords, countWordNode);
                    fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    fixWords[StableData.INT_ZERO].append(countWordNode);
                    continue Here;
                              }
                    if (compare == StableData.INT_TWO) {

                    countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
                              , outputList, stringBuilder, wordsForest, fixWords, posController);
                    continue Here;
                              }
                    if (compare == StableData.INT_THREE) {
                    addFixWords(charPosition, inputString, fixWords);
                    countInputStringLength= nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
                              , wordsForest, stringBuilder, fixWords, posController);
                    continue Here;
                              }
                    if (compare == StableData.INT_FOUR) {
                    addFixWords(charPosition, inputString, fixWords);
                    countInputStringLength= nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
                              , wordsForest, fixWords, posController);
                              }}
                    return outputList;
          }
public void addFixWords(int charPosition, String inputString, StringBuilder[] fixWords) {
                    fixWords[StableData.INT_ONE].delete(StableData.INT_ZERO, fixWords[StableData.INT_ONE].length());
          if (charPosition+ StableData.INT_EIGHT < inputString.length()) {
                              fixWords[StableData.INT_ONE].append(inputString.substring(charPosition+ StableData.INT_THREE
                    , charPosition+ StableData.INT_EIGHT));
                              return;
                    }
                    fixWords[StableData.INT_ONE].append(inputString.substring(charPosition+ StableData.INT_THREE
                    , inputString.length()));
          }
public Map<String, String> getEnToCn() {
                    return fHMMList.getEnToCn();
          }
public Map<String, String> getCnToEn() {
                    return fHMMList.getCnToEn();
          }
public Map<String, WordFrequency> getWordFrequencyMap(List<String> sets) throws IOException {
          Map<String, WordFrequency> map= new ConcurrentHashMap<>();
                    Iterator <String> iterator= sets.iterator();
                    Here:
                              while(iterator.hasNext()){
                    String setOfi= iterator.next();
          if (map.containsKey(setOfi)) {
                    WordFrequency wordFrequency= map.get(setOfi);
                    wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
          Map.put(setOfi, wordFrequency);
                    continue Here;
                    }
                    WordFrequency wordFrequency= new WordFrequency();
                    wordFrequency.setFrequency(StableData.INT_ONE);
                    wordFrequency.setWord(setOfi);
          Map.put(setOfi, wordFrequency);
                              }
                    return map;
          }
public List<WordFrequency> sortWordFrequencyMap(Map<String, WordFrequency> map) throws IOException
          {
          List<WordFrequency> list= quick6DLuoYaoguangSort.frequencyWordMapToList(map);
                    quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(list, StableData.INT_ZERO
                    , list.size() - StableData.INT_ONE);
                    return list;
          }
public List<WordFrequency> getWordFrequency(List<String> sets) throws IOException {
                    return sortWordFrequencyMap(getWordFrequencyMap(sets));
          }
public Map<Integer, WordFrequency> getWordFrequencyByReturnSortMap(List<String> sets) throws IOException {
                    return sortWordFrequencyMapToSortMap(getWordFrequencyMap(sets));
          }
public Map<Integer, WordFrequency> sortWordFrequencyMapToUnsortMap(Map<String, WordFrequency> map){
          Map<Integer, WordFrequency> listMap= quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
                    return listMap;
          }
public Map<Integer, WordFrequency> sortWordFrequencyMapToSortMap(Map<String, WordFrequency> map){
          Map<Integer, WordFrequency> listMap= quick6DLuoYaoguangSort.frequencyWordMapToMap(map);
                    quick6DLuoYaoguangSort.quick6DLuoYaoGuangSortWordFrequency(listMap, StableData.INT_ZERO
                    , listMap.size() - StableData.INT_ONE);
                    return listMap;
          }
```

```
public String[] parserEnglishString(String englishString) {
                    String[] words= englishString.replaceAll(StableData.NLP_SPASE_REP,
StableData.SPACE_STRING).split(StableData.SPACE_STRING);
          if(StableData.INT_ZERO == words.length ) {
                              return new String[] {
                    StableData.SPACE_STRING};
                    }
                    return words;
          }
public Map<String, String> getPosEnToCn() {
                    return fHMMList.getPosEnToCn();
          }
public Map<String, String> getPosEnToEn() {
                    return fHMMList.getPosEnToEn();
          }
public Map<String, String> getPosCnToCn() {
                    return fHMMList.getPosCnToCn();
          }
public Map<String, String> getFullEnToCn() {
                    return fHMMList.getFullEnToCn();
          }
public Map<String, String> getFullCnToEn() {
                    return fHMMList.getFullCnToEn();
          }
public Map<String, WordFrequency> parserMixStringByReturnFrequencyMap(String mixedString) {
                    mixedString += StableData.SPACE_STRING;
          Map<String, String> wordsForest= fHMMList.getPosCnToCn();
          Map<String, WordFrequency> outputList= new ConcurrentHashMap<>();
          Map<Long, FMHMMNode> forestRoots= fHMMList.getMap();
                    //.getRoot();
                    int inputStringLength= mixedString.length();
                    int forestDepth= StableData.INT_ZERO;
                    int countInputStringLength;
                    StringBuilder[] fixWords= new StringBuilder[StableData.INT_TWO];
                    fixWords[StableData.INT_ZERO]= new StringBuilder();
                    fixWords[StableData.INT_ONE]= new StringBuilder();
                    StringBuilder stringBuilder= new StringBuilder();
                    int find= StableData.INT_ZERO;
                    Here:
                              for (int charPosition= StableData.INT_ZERO;
                    charPosition < inputStringLength;
                    charPosition
+= (countInputStringLength == StableData.INT_ZERO ? StableData.INT_ONE : countInputStringLength)) {

                    //luan ma
          if(mixedString.charAt(charPosition) < StableData.INT_ONE_TWO_EIGHT && charPosition < mixedString.length()
                              - StableData.INT_ONE){
          if(find == StableData.INT_ZERO) {
                              fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                    }
                    fixWords[StableData.INT_ZERO].append(mixedString.charAt(charPosition));
                    countInputStringLength= StableData.INT_ONE;
                    find= StableData.INT_ONE;
                    continue Here;
                    }
          if(find == StableData.INT_ONE) {
                    find= StableData.INT_ZERO;
                    Iterator<String> it =
fHMMList.englishStringToWordsList(fixWords[StableData.INT_ZERO].toString()).iterator();
                    while(it.hasNext()) {
                              String temp=it.next();if (outputList.containsKey(temp)) {
                    WordFrequency wordFrequency= outputList.get(temp);
wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
outputList.put(temp, wordFrequency);
                              } else {
                    WordFrequency wordFrequency= new WordFrequency();
wordFrequency.setFrequency(StableData.INT_ONE);
wordFrequency.setWord(temp);
outputList.put(temp, wordFrequency);
                                }}
                        fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                        }
                        stringBuilder.delete(StableData.INT_ZERO, stringBuilder.length());
                        stringBuilder= neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString
                                .charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth
                                , charPosition+ StableData.INT_ONE);
                        String countWordNode= stringBuilder.toString();
                        int compare= countInputStringLength= countWordNode.length();
            if (compare == StableData.INT_TWO) {
                        countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoCharForMap(countInputStringLength
, outputList, stringBuilder, wordsForest, fixWords, posController);
                        continue Here;
                        }
            if (compare == StableData.INT_THREE) {
                        addFixWords(charPosition, mixedString, fixWords);
                        countInputStringLength= nlpController.doPOSAndEMMCheckOfThreeForMap(countInputStringLength, outputList
, wordsForest, stringBuilder, fixWords, posController);
                        continue Here;
                        }
            if (compare == StableData.INT_ONE) {
            if (outputList.containsKey(countWordNode)) {
                                WordFrequency wordFrequency= outputList.get(countWordNode);
                                wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                                outputList.put(countWordNode, wordFrequency);
                        } else {
                                WordFrequency wordFrequency= new WordFrequency();
                                wordFrequency.setFrequency(StableData.INT_ONE);
                                wordFrequency.setWord(fixWords[StableData.INT_ZERO].toString());
                                outputList.put(countWordNode, wordFrequency);
                        }
                        fixWords[StableData.INT_ZERO].delete(StableData.INT_ZERO, fixWords[StableData.INT_ZERO].length());
                        fixWords[StableData.INT_ZERO].append(countWordNode);
                        continue Here;
                        }
            if (compare == StableData.INT_FOUR) {
                        addFixWords(charPosition, mixedString, fixWords);
                        countInputStringLength= nlpController.doSlangCheckForMap(countInputStringLength, outputList, stringBuilder
, wordsForest, fixWords, posController);
                        }}
                        return outputList;
            }
public void studyNewWord(String study, String token, String posStudy) {
                        //learn new wordFMHMMNode fFHMMNode= forestRoots.get(Long.valueOf(study.charAt(StableData.INT_ZERO)));
            Map<String, Integer> map;
            if(null== fFHMMNode) {
                                fFHMMNode= new FMHMMNode();
                        Map= new ConcurrentHashMap<>();
                        }else {
                        Map= fFHMMNode.getNext();
                        }
            Map.put(token, StableData.INT_ONE);
                        fFHMMNode.setNext(map);
                        forestRoots.put(Long.valueOf(study.charAt(StableData.INT_ZERO)), fFHMMNode);
                        //learn new pos fHMMList.studyNewPos(study+token, posStudy);
            }
            @Override
public Map<String, String> getStudyPos() {
                        return fHMMList.getStudyPos();
            }}
-----------------------------------------------------------------------------------------------------------------------
package org.tinos.engin.utils;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
```

```
public class WordFrequencyUtil {
public static void WordFrequencyFindCheck(Map<String, WordFrequency> outputList,StringBuilder[] fixWords) {
                String string= fixWords[StableData.INT_ZERO].toString();
        if (outputList.containsKey(string)) {
                        WordFrequency wordFrequency= outputList.get(string);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(string, wordFrequency);
                        return;
                }
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(string);
                outputList.put(string, wordFrequency);
        }
public static void WordFrequencyCompareCheck(Map<String, WordFrequency> outputList, StringBuilder[] fixWords, String
countWordNode) {
        if (outputList.containsKey(countWordNode)) {
                        WordFrequency wordFrequency=outputList.get(countWordNode);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(countWordNode, wordFrequency);
                        return;
                }
                WordFrequency wordFrequency=new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(fixWords[StableData.INT_ZERO].toString());
                outputList.put(countWordNode,wordFrequency);
        }}
------------------------------------------------------------------------------------------------------------------------------
package org.tinos.engin.utils;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
public class WordForestUtil {
public static void wordsForestNotContainsKey(Map<String, WordFrequency> outputList
                        , String countWordNode, StringBuilder[] prefixWord) {
                String string= String.valueOf(countWordNode.charAt(StableData.INT_ZERO));
        if (outputList.containsKey(string)) {
                        WordFrequency wordFrequency= outputList.get(string);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(string, wordFrequency);
                        prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO,
prefixWord[StableData.INT_ZERO].length());
                        prefixWord[StableData.INT_ZERO].append(countWordNode.charAt(StableData.INT_ZERO));
                        return;
                }
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(string);
                outputList.put(string, wordFrequency);
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(countWordNode.charAt(StableData.INT_ZERO));
        }
public static void prefixWordEqualZero(Map<String, WordFrequency> outputList,String countWordNode
                        , StringBuilder[] prefixWord) {
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(countWordNode);
        if (outputList.containsKey(countWordNode)) {
                        WordFrequency wordFrequency= outputList.get(countWordNode);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(countWordNode, wordFrequency);
                        return;
                }
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(countWordNode);
                outputList.put(countWordNode, wordFrequency);
        }public static void wordsForestContainsKey(Map<String, WordFrequency> outputList,String countWordNode
                        , StringBuilder[] prefixWord) {
                prefixWord[StableData.INT_ZERO].delete(StableData.INT_ZERO, prefixWord[StableData.INT_ZERO].length());
                prefixWord[StableData.INT_ZERO].append(countWordNode);
        if (outputList.containsKey(countWordNode)) {
                        WordFrequency wordFrequency= outputList.get(countWordNode);
                        wordFrequency.setFrequency(wordFrequency.getFrequency()+ StableData.INT_ONE);
                        outputList.put(countWordNode, wordFrequency);
                        return;
                }
                WordFrequency wordFrequency= new WordFrequency();
                wordFrequency.setFrequency(StableData.INT_ONE);
                wordFrequency.setWord(countWordNode);
                outputList.put(countWordNode, wordFrequency);
        }}
------------------------------------------------------------------------------------------------------------------------------------
package org.tinos.sensing.ortho.fhmm;
import java.io.IOException;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
public interface SensingMap{
                void initLenovoMap(Analyzer analyzer) throws IOException;
public Map<String, Object> getLenovoMap();
public void setLenovoMap(Map<String, Object> lenovoMap);
}
------------------------------------------------------------------------------------------------------------------------------------
package org.tinos.sensing.ortho.fhmm.imp;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.sensing.ortho.fhmm.SensingMap;
public class SensingMapImp implements SensingMap{
        private Map<String, Object> lenovoMap;
        @Override
public Map<String, Object> getLenovoMap() {
                return this.lenovoMap;
        }
        @Override
public void setLenovoMap(Map<String, Object> lenovoMap) {
                this.lenovoMap= lenovoMap;
        }
        @Override
public void initLenovoMap(Analyzer analyzer) throws IOException {
                lenovoMap= new HashMap<>();
        Map<String, String> CnToEnMap= analyzer.getFullCnToEn();
        Map<String, String> EnToCnMap= analyzer.getEnToCn();
                Iterator<String> iterator= CnToEnMap.keySet().iterator();
                Here:
                        while(iterator.hasNext()) {
                String word= iterator.next();
        if(!CnToEnMap.containsKey(word)) {
                continue Here;
                }
                        if(!EnToCnMap.containsKey(CnToEnMap.get(word))) {
                lenovoMap.put(word, word);
                continue Here;
                }
                lenovoMap.put(word, EnToCnMap.get(CnToEnMap.get(word)));
                        }}}
------------------------------------------------------------------------------------------------------------------------------------
package org.tinos.test;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.engine.base.translator.Translator;
import org.tinos.engine.base.translator.imp.TranslatorImp;
//import org.tinos.freetts.thread.read.ReadEnglish;
import org.tinos.view.obj.Verbal;
//import timeProcessor.TimeCheck;
public class DemoTSLT {
        @SuppressWarnings("unused")
public static void main(String[] args) throws IOException {
                Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
                analyzer.init();
        Map<String, String> ce= analyzer.getCnToEn();
        Map<String, String> ec= analyzer.getEnToCn();
        Map<String, String> poscc= analyzer.getPosCnToCn();
        Map<String, String> posec= analyzer.getPosEnToCn();
        Map<String, String> posee= analyzer.getPosEnToEn();
        Map<String, String> fce= analyzer.getFullCnToEn();
        Map<String, String> fec= analyzer.getFullEnToCn();
                System.out.println("输入");
                String v= "如流枫之回雪，若浣花洗月";
                ////
                //String v= "数据一直在更新中";
                //
                String v= "中国正在崛起的道路上奔跑";
                //
                String v= "我一直在奔跑，我需要一双翅膀！";
                // String v= "他也一直在奔跑，他同样需要一双翅膀！";
                System.out.println(v);
                Translator ts= new TranslatorImp();
                ts.init(analyzer);
                //
                TimeCheck t= new TimeCheck();
                // t.begin();
        List<Verbal> verbals= ts.index(analyzer, v);
                String ch= ts.getChineseSentenseFromVerbalList(verbals);
                // t.end();
                System.out.println("中文");
                System.out.println(ch);
                for(int i=0;
                i<verbals.size();
                i++) {
                        System.out.print(verbals.get(i).getChinese());
                        System.out.print(verbals.get(i).getEnglish());
                        System.out.print(verbals.get(i).getExplain());
                        System.out.println(verbals.get(i).getPartOfSpeech());
                }
        List<Verbal> verbalsFix= ts.fixPos(verbals);
                String en= ts.getEnglishSentenseFromVerbalFixList(verbalsFix);
                System.out.println("");
                System.out.println("英文");
                System.out.println(en);
                for(int i=0;
                i<verbalsFix.size();
                i++) {
                        System.out.print(verbalsFix.get(i).getChinese());
                        System.out.print(verbalsFix.get(i).getEnglish());
                        System.out.print(verbalsFix.get(i).getExplain());
                        System.out.println(verbalsFix.get(i).getPartOfSpeech());
                }
                System.out.println("");
                System.out.println("中文解释");
                for(int i=0;
                i<verbals.size();
                i++) {
                        System.out.print(verbals.get(i).getExplain()==null?verbals.get(i).getChinese():verbals.get(i).getExplain());
                }
                // t.duration();
```

```
                    //
                    ReadEnglish readEnglish= new ReadEnglish();
                    // readEnglish.setPreReadText(en);
                    // readEnglish.start();
          }}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.base.translator;
import java.io.IOException;
import java.util.List;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.view.obj.Verbal;
public interface Translator{
          void init(Analyzer analyzer) throws IOException;
          String EnglishStringToChineseString(Analyzer analyzer, String EnglishString);
          String ChineseStringToEnglishString(Analyzer analyzer, String ChineseString);
          String MixedStringToChineseString(Analyzer analyzer, String key);
          String ChineseStringToEnglishStringWithPOS(Analyzer analyzer, String v);
          List<Verbal> index(Analyzer analyzer, String mixedString);
                    List<Verbal> fixPos(List<Verbal> verbals);
          String getChineseSentenseFromVerbalList(List<Verbal> verbals);
          String getEnglishSentenseFromVerbalFixList(List<Verbal> verbalsFix);
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.engine.base.translator.imp;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.CopyOnWriteArrayList;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.base.translator.Translator;
import org.tinos.view.obj.Verbal;
import org.tinos.view.stable.StableData;
public class TranslatorImp implements Translator{
public Map<String, String> poscc;
public Map<String, String> posec;
public Map<String, String> posee;
public Map<String, String> etc;
public Map<String, String> cte;
public Map<String, String> fulletc;
public Map<String, String> fullcte;
public void init(Analyzer analyzer) throws IOException {
                    posec= analyzer.getPosEnToCn();
                    posee= analyzer.getPosEnToEn();
                    poscc= analyzer.getPosCnToCn();
                    etc= analyzer.getEnToCn();
                    cte= analyzer.getCnToEn();
                    fulletc= analyzer.getFullEnToCn();
                    fullcte= analyzer.getFullCnToEn();
          }
public String EnglishStringToChineseString(Analyzer analyzer, String EnglishString) {
                    String[] nodes= analyzer.parserEnglishString(EnglishString);
                    StringBuilder sb= new StringBuilder();
                    for(String temp:nodes) {
                              char[] caseTemp= temp.toCharArray();
                              caseTemp[StableData.INT_ONE]=
String.valueOf(caseTemp[StableData.INT_ONE]).toUpperCase().charAt(StableData.INT_ONE);if(etc.containsKey(temp)) {

                    sb.append(etc.get(temp).split(StableData.NLP_DOT)[StableData.INT_ONE]);
                              }else if(fulletc.containsKey(String.valueOf(caseTemp))){
                    sb.append(fulletc.get(String.valueOf(caseTemp)));
                              }else {
                    sb.append(temp);
                              }
                              sb.append(StableData.SPACE_STRING);
                    }
                    return sb.toString();
```

```
        }
public String ChineseStringToEnglishString(Analyzer analyzer, String ChineseString) {
        List<String> nodes= analyzer.parserString(ChineseString);
                StringBuilder sb= new StringBuilder();
                Iterator<String> it= nodes.iterator();
                while(it.hasNext()) {
                        String temp= it.next();if(cte.containsKey(temp)) {
                sb.append(cte.get(temp));
                        }else if(fullcte.containsKey(temp)){
                sb.append(fullcte.get(temp));
                        }else {
                sb.append(temp);
                        }
                        sb.append(StableData.SPACE_STRING);
                }
                return sb.toString();
        }
public String MixedStringToChineseString(Analyzer analyzer, String mixedString) {
        List<String> nodes= analyzer.parserMixedString(mixedString.toLowerCase());
                StringBuilder sb= new StringBuilder();
                Iterator<String> it= nodes.iterator();
                while(it.hasNext()) {
                        String temp= it.next();if(poscc.containsKey(temp)) {
                sb.append(temp);
                        }else {

                String[] strings= analyzer.parserEnglishString(temp);
                for(String string:strings) {
        if(string == null || string.length() < StableData.INT_ONE) {
                        string= StableData.EMPTY_STRING;
                }
        if(etc.containsKey(string)) {
                        sb.append(etc.get(string).split(StableData.NLP_DOT)[StableData.INT_ZERO]);
                }else if(fulletc.containsKey(string)){
                        sb.append(fulletc.get(string));
                }else {
                        sb.append(temp);
                }}}}
                return sb.toString();
        }
public String ChineseStringToEnglishStringWithPOS(Analyzer analyzer, String ChineseString) {
        List<String> nodes= analyzer.parserString(ChineseString);
                StringBuilder sb= new StringBuilder();
                Iterator<String> it= nodes.iterator();
                while(it.hasNext()) {
                        String temp= it.next();if(cte.containsKey(temp)) {

                sb.append(cte.get(temp));
                        }else if(fullcte.containsKey(temp)){
                sb.append(fullcte.get(temp));
                        }else {
                sb.append(temp);
                        }
                if(poscc.containsKey(temp)) {
                sb.append(StableData.NLP_SYMBO_SLASH);
                sb.append(poscc.get(temp));
                        }
                        sb.append(StableData.SPACE_STRING);
                }
                return sb.toString();
        }
public List<Verbal> index(Analyzer analyzer, String mixedString) {
        List<Verbal> verbals= new CopyOnWriteArrayList<>();
        List<String> nodes= analyzer.parserMixedString(mixedString.toLowerCase());
                Iterator<String> it= nodes.iterator();
                while(it.hasNext()) {
                        String word= it.next();
```

```
                    Verbal verbal= new Verbal();if(poscc.containsKey(word)) {
            verbal.setChinese(word);
            verbal.setPartOfSpeech(poscc.get(word));
    if(cte.containsKey(word)) {
            verbal.setEnglish(cte.get(word));
    if(etc.containsKey(cte.get(word))) {
                    verbal.setExplain(etc.get(cte.get(word)));
            }}else if(fullcte.containsKey(word)){
                    verbal.setEnglish(fullcte.get(word));
                    verbal.setExplain(fullcte.get(word));
            }}else if(posee.containsKey(word)) {
                    verbal.setEnglish(word);if(fulletc.containsKey(word)) {
verbal.setChinese(fulletc.get(word));
                    }
            if(poscc.containsKey(fulletc.get(word))) {
verbal.setPartOfSpeech(poscc.get(fulletc.get(word)));
                    }
            if(etc.containsKey(word)) {
verbal.setExplain(etc.get(word));
                    }}else {
verbal.setEnglish(word);
verbal.setChinese(fulletc.get(word));
verbal.setPartOfSpeech(StableData.NLP_NULL);
verbal.setExplain(StableData.NLP_NULL);
                    }
                    verbals.add(verbal);
            }
            return verbals;
    }
public List<Verbal> fixPos(List<Verbal> verbals) {
            for(int i= 0;
            i < verbals.size();
            i++) {
            if(verbals.get(i).getPartOfSpeech() != null) {
                    if(verbals.get(i).getPartOfSpeech().contains(StableData.NLP_ZI_DONG)){
    if(!verbals.get(i).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
            if(i - StableData.INT_ONE > StableData.INT_ZERO && verbals.get(i - StableData.INT_ONE)
    .getEnglish().contains(StableData.NLP_ENGLISH_OF)) {
Verbal temp= verbals.get(i);
temp.setPartOfSpeech(StableData.NLP_CI_DONG_MING);
String english= temp.getEnglish().replace(StableData.SPACE_STRING, StableData.EMPTY_STRING);if(english.charAt(english.length() -
StableData.INT_ONE) == StableData.NLP_CHAR_E) {
    english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING
+ english.substring(StableData.INT_ZERO, english.length() - StableData.INT_ONE)
+ StableData.NLP_ENGLISH_ING;
}else {
                    english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING+ english+
StableData.NLP_ENGLISH_ING;
}
temp.setEnglish(english);
                    }else if(verbals.get(i+ StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
if(i-
            StableData.INT_ONE >= StableData.INT_ZERO && !verbals.get(i-
            StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
            if(!verbals.get(i - StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_DAI)){
    if(verbals.get(i-
            StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_JIE)){
            if(verbals.get(i-
            StableData.INT_ONE).getChinese().contains(StableData.NLP_ZI_ZAI)){
    if(verbals.get(i+ StableData.INT_ONE).getChinese().contains(StableData.NLP_ZI_ZHONG)){
            Verbal temp= verbals.get(i+ StableData.INT_ONE);
            temp.setEnglish(StableData.NLP_ENGLISH_STATUS);
            }}}
            Verbal temp= verbals.get(i);
            temp.setPartOfSpeech(StableData.NLP_CI_DONG_MING);
            String english= temp.getEnglish().replace(StableData.SPACE_STRING, StableData.EMPTY_STRING);
    if(english.charAt(english.length()-StableData.INT_ONE) == StableData.NLP_CHAR_E) {
```

```
                        english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING
+ english.substring(StableData.INT_ZERO, english.length()
-
StableData.INT_ONE)
+ StableData.NLP_ENGLISH_ING;
                }else {
                        english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING+ english
+ StableData.NLP_ENGLISH_ING;
                }
                temp.setEnglish(english);
        }}
if(verbals.get(i+ StableData.INT_TWO).getPartOfSpeech().contains(StableData.NLP_ZI_DONG)){
        if(!verbals.get(i+ StableData.INT_TWO).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
        if(verbals.get(i+ StableData.INT_THREE).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
                        Verbal temp= verbals.get(i+ StableData.INT_TWO);
                        temp.setPartOfSpeech(StableData.NLP_CI_DONG_MING);
                        String english= temp.getEnglish().replace(StableData.SPACE_STRING,
StableData.EMPTY_STRING);if(english.charAt(english.length()-
                StableData.INT_ONE) == StableData.NLP_CHAR_E) {

                english= StableData.NLP_ENGLISH_OF+ StableData.SPACE_STRING+ StableData.NLP_ENGLISH_THE+
StableData.SPACE_STRING
+ english.substring(StableData.INT_ZERO, english.length()-StableData.INT_ONE)
+ StableData.NLP_ENGLISH_ING;
                        }else {

                english= StableData.NLP_ENGLISH_OF+ StableData.SPACE_STRING+ StableData.NLP_ENGLISH_THE+
StableData.SPACE_STRING +
                        english+ StableData.NLP_ENGLISH_ING;
                        }
                        temp.setEnglish(english);
                }}}}else if(i - StableData.INT_ONE >= StableData.INT_ZERO && verbals.get(i -
StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_JIE)){
                        Verbal temp= verbals.get(i);
                        temp.setPartOfSpeech(StableData.NLP_CI_DONG_MING);
                        String english= temp.getEnglish().replace(StableData.SPACE_STRING,
StableData.EMPTY_STRING);if(english.charAt(english.length() - StableData.INT_ONE) == StableData.NLP_CHAR_E) {
                english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING+ english.substring(StableData.INT_ZERO
                        , english.length() - StableData.INT_ONE)+ StableData.NLP_ENGLISH_ING;
                        }else {
                english= StableData.NLP_ENGLISH_THE+ StableData.SPACE_STRING+ english+ StableData.NLP_ENGLISH_ING;
                        }
                        temp.setEnglish(english);
                }else if(i-
                StableData.INT_ONE >= StableData.INT_ZERO && verbals.get(i
                        -StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_FU)){
                        Verbal temp= verbals.get(i);
                        temp.setPartOfSpeech(StableData.NLP_CI_DONG_MING);
                        String english= temp.getEnglish().replace(StableData.SPACE_STRING,
StableData.EMPTY_STRING);if(english.charAt(english.length() - StableData.INT_ONE) == StableData.NLP_CHAR_E) {
                english= english.substring(StableData.INT_ZERO, english.length() - StableData.INT_ONE)
+ StableData.NLP_ENGLISH_ING;
                        }else {
                                english += StableData.NLP_ENGLISH_ING;
                        }
                        temp.setEnglish(english);
                }}}else if(verbals.get(i).getPartOfSpeech().contains(StableData.NLP_ZI_MING)){
                if(i - StableData.INT_ONE >= StableData.INT_ZERO && verbals.get(i -
StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_FU_SHU)){
                Verbal temp= verbals.get(i);
                String english= temp.getEnglish().replace(StableData.SPACE_STRING, StableData.EMPTY_STRING);
        if(english.charAt(english.length()-1) == StableData.NLP_CHAR_H || english.charAt(english.length()-1)
                        == StableData.NLP_CHAR_S) {
                english += StableData.NLP_ENGLISH_ES;
                }else {
                english += StableData.NLP_ENGLISH_S;
                }
```

```
                    temp.setEnglish(english);
                              }}else if(verbals.get(i).getPartOfSpeech().contains(StableData.NLP_CI_FU)){
                              if(i - StableData.INT_ONE >= StableData.INT_ZERO && (verbals.get(i-StableData.INT_ONE)
                              .getPartOfSpeech().contains(StableData.NLP_ZI_DAI)
                              ||verbals.get(i - StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_MING))){
          if(i-StableData.INT_ONE >= StableData.INT_ZERO&&verbals.get(i
-
StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_复)
||(verbals.get(i-StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_单) &&
                    verbals.get(i-StableData.INT_ONE).getPartOfSpeech().contains(StableData.NLP_ZI_一))){
                    if(verbals.get(i).getEnglish().contains(StableData.NLP_HAVE_HAS)) {
                    Verbal temp= verbals.get(i);
String english= temp.getEnglish().replace(StableData.NLP_HAVE_HAS, StableData.NLP_HAVE);
temp.setEnglish(english);
                              }}else {
if(verbals.get(i).getEnglish().contains(StableData.NLP_HAVE_HAS)) {
          Verbal temp= verbals.get(i);
          String english= temp.getEnglish().replace(StableData.NLP_HAVE_HAS, StableData.NLP_HAS);
          temp.setEnglish(english);
}}}
          if(i-StableData.INT_TWO>= StableData.INT_ZERO && (verbals.get(i-StableData.INT_TWO).getPartOfSpeech()
                              .contains(StableData.NLP_ZI_DAI) || verbals.get(i-StableData.INT_TWO).getPartOfSpeech()
                              .contains(StableData.NLP_ZI_MING))){
          if(verbals.get(i-StableData.INT_TWO).getPartOfSpeech().contains(StableData.NLP_ZI_复)
||(verbals.get(i-StableData.INT_TWO).getPartOfSpeech().contains(StableData.NLP_ZI_ 单 ) &&verbals.get(i-
StableData.INT_TWO).getPartOfSpeech().contains(StableData.NLP_ZI_一))){
                    if(verbals.get(i).getEnglish().contains(StableData.NLP_HAVE_HAS)) {
                    Verbal temp= verbals.get(i);
String english= temp.getEnglish().replace(StableData.NLP_HAVE_HAS, StableData.NLP_HAVE);
temp.setEnglish(english);
                              }}else {
if(verbals.get(i).getEnglish().contains(StableData.NLP_HAVE_HAS)) {
          Verbal temp= verbals.get(i);
          String english= temp.getEnglish().replace(StableData.NLP_HAVE_HAS, StableData.NLP_HAS);
          temp.setEnglish(english);
}}}}
                              }}
                    return verbals;
          }
public String getChineseSentenseFromVerbalList(List<Verbal> verbals) {
                    StringBuilder sb= new StringBuilder();
                    for(int i= 0;
                    i < verbals.size();
                    i++) {
                              sb.append(verbals.get(i).getChinese() == null ? StableData.EMPTY_STRING : verbals.get(i).getChinese());
                    }
                    return sb.toString().replaceAll(StableData.NLP_SPASE_REP, StableData.SPACE_STRING);
          }
public String getEnglishSentenseFromVerbalFixList(List<Verbal> verbalsFix) {
                    StringBuilder sb= new StringBuilder();
                    for(int i= 0;
                    i<verbalsFix.size();
                    i++) {
                              sb.append(verbalsFix.get(i).getEnglish()==null ? StableData.EMPTY_STRING :
verbalsFix.get(i).getEnglish().toLowerCase());
                              sb.append(StableData.SPACE_STRING);
                    }
                    return sb.toString().replaceAll(StableData.NLP_SPASE_REP, StableData.SPACE_STRING);
          }}
-------------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.test;
import java.io.IOException;
import java.util.Iterator;
import java.util.Map;
import org.tinos.emotion.engine.LenovoInit;
import org.tinos.emotion.estimation.EmotionSample;
public class LenovoTest{
```

```
public static void main(String[] argv) throws IOException {
                //init  //20210727 采样数据仅保存一次，之后相同全部删除
                String text= "夏天的太阳像个大火球似的火辣辣地照射着大地，似乎要散发出"
        + "全部的力量。它炙烤着大地，晒红了行人的脸膛，晒得大树不敢有丝毫"
        + "摆动。如此热的天，我们更是盼望太阳能早点下山。盼啊盼，盼得太阳不"
        + "好意思了，慢慢退出天空。夏天的傍晚，我可喜欢啦！女孩子穿着漂亮的"
        + "泳装，神采飞扬。男孩子全身除了一条小裤衩，都裸露着，浑身光溜溜像"
        + "条泥鳅。这时，小河成了我们的乐园。到了河边，就一头扎进去。有的打"
        + "“狗刨”，有的“漂长江”，有的“蝶泳”，有的像泥鳅一样在河里穿梭来往"
        + "。我最喜欢打水仗了，你泼我，我泼你。刹那时水花四溅，连抹把脸都顾不"
        + "上。我们有时也会闹不愉快，被水淋得透不过气来，会哭，但一会儿便充满"
        + "了笑声。河里溅起朵朵雪白的浪花，小伙伴的欢声笑语和击水的哗哗声交织"
        + "在一起，就像一曲动听的田园交响曲。每当这时，我快乐极了，迷人的小河"
        + "，带给我夏天最美的享受\r\n"+ "。";
                LenovoInit lenovoInit= new LenovoInit();
                lenovoInit.init(text);
        Map<String, EmotionSample> environmentSampleMap= lenovoInit.getEnvironmentInit().getEmotionSampleMap();
        Map<String, Object> lenovo= lenovoInit.getSensingMap().getLenovoMap();
                //reduce
                System.out.println("环 境：");
                Iterator<String> Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getDistinction()) {
        if(lenovo.containsKey(emotionSample.getDistinction())) {
                System.out.print(lenovo.get(emotionSample.getDistinction()).toString()+" ");
                }else {
                System.out.print(emotionSample.getDistinction()+" ");
                }}}
                System.out.println("");
                System.out.println("动机联想：");
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getMotivation()) {
        if(lenovo.containsKey(emotionSample.getMotivation())) {
                System.out.print(lenovo.get(emotionSample.getMotivation()).toString()+" ");
                }else {
                System.out.print(emotionSample.getMotivation()+" ");
                }}}
                System.out.println("");
                System.out.println("倾向探索：" );
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getTrending()) {
        if(lenovo.containsKey(emotionSample.getTrending())) {
                System.out.print(lenovo.get(emotionSample.getTrending()).toString()+" ");
                }else {
                System.out.print(emotionSample.getTrending()+" ");
                }}}
                //reduce System.out.println("");
                System.out.println("决策挖掘：");
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getPrediction()) {
        if(lenovo.containsKey(emotionSample.getPrediction())) {
                System.out.print(lenovo.get(emotionSample.getPrediction()).toString()+" ");
                }else {
                System.out.print(emotionSample.getPrediction()+" ");
                }}}}
}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.test;
```

```
import java.io.IOException;
import java.util.Iterator;
import java.util.Map;
import org.tinos.emotion.engine.EnvironmentInit;
import org.tinos.emotion.estimation.EmotionSample;
public class EnvironmentTest{
public static void main(String[] argv) throws IOException {
                    //init  //20210727 采样数据仅保存一次，之后相同全部删除
                    String text= "关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。\r\n"+ "关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而\r\n" +
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。\r\n"+ "替代物，本身也是有强烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗\r\n" +
"脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易\r\n"+ "感染群体进行教育和传授方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进\r\n"+ "化的产物，是与生俱来的。上瘾是一种外力干涉造成的依赖。上瘾的级别有很多种。医学有相关严谨\r\n"+ "的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活方式的选择。很多时候\r\n" +
"环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触\r\n" +
"成瘾源。当环境无法改变的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，\r\n"+ "很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。\r\n" +
"一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境\r\n"+
"因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。\r\n"+
"如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的。";
                    EnvironmentInit environmentInit= new EnvironmentInit();
                    environmentInit.init(text);
          Map<String, EmotionSample> environmentSampleMap= environmentInit.getEmotionSampleMap();
                    //reduce
                    System.out.println("环 境：");
                    Iterator<String> Iterator= environmentSampleMap.keySet().iterator();
                    while(Iterator.hasNext()) {
                              String word= Iterator.next();
                              EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getDistinction()) {
                    System.out.print(emotionSample.getDistinction()+" ");
                              }}
                    System.out.println("");
                    System.out.println("动 机：");
                    Iterator= environmentSampleMap.keySet().iterator();
                    while(Iterator.hasNext()) {
                              String word= Iterator.next();
                              EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getMotivation()) {
                    System.out.print(emotionSample.getMotivation()+" ");
                              }}
                    System.out.println("");
                    System.out.println("倾 向：" );
                    Iterator= environmentSampleMap.keySet().iterator();
                    while(Iterator.hasNext()) {
                              String word= Iterator.next();
                              EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getTrending()) {
                    System.out.print(emotionSample.getTrending()+" ");
                              }}
                    //reduce System.out.println("");
                    System.out.println("决 策：");
                    Iterator= environmentSampleMap.keySet().iterator();
                    while(Iterator.hasNext()) {
                              String word= Iterator.next();
                              EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getPrediction()) {
                    System.out.print(emotionSample.getPrediction()+" ");
                              }}}}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.test;
import java.io.IOException;
import org.tinos.emotion.engine.EmotionInit;
public class EmotionTest{
public static void main(String[] argv) throws IOException {
                    String text= "  //20210727 采样数据仅保存一次，之后相同全部删除";
                    EmotionInit emotionInit= new EmotionInit();
```

```
                emotionInit.init(text);
                //reduce
                double positiveCount= emotionInit.getPositiveCount();
                double negativeCount= emotionInit.getNegativeCount();
                double totalCount= emotionInit.getTotalCount();
                System.out.println(" 正 面 数 :"+ positiveCount);
                System.out.println(" 负 面 数 :"+ negativeCount);
        if(positiveCount == 0) {
                        positiveCount= 1;
                }
        if(negativeCount == 0) {
                        negativeCount= 1;
                }
                double adjRatio= Math.abs(positiveCount/negativeCount-negativeCount/positiveCount);
                System.out.println("渲染比率:"+ adjRatio);
                double phychologicRatio= (positiveCount+ negativeCount)/totalCount;
                System.out.println("情绪比率:"+ phychologicRatio);
                double infectionRatio= adjRatio*phychologicRatio;
                System.out.println("感染比率:"+ infectionRatio);
        }}
------------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.ortho.fhmm;
import java.io.IOException;
import java.util.Map;
public interface EmotionMap{
                void initMotivationMap() throws IOException;
        void initPositiveMap() throws IOException;
        void initNegativeMap() throws IOException;
        void initTrendingMap() throws IOException;
        void initPredictionMap() throws IOException;
        void initDistinctionMap() throws IOException;
public Map<String, Object> getPositiveMap();
public void setPositiveMap(Map<String, Object> positiveMap);
public Map<String, Object> getNegativeMap();
public void setNegativeMap(Map<String, Object> negativeMap);
public Map<String, Object> getMotivationMap() ;
public void setMotivationMap(Map<String, Object> motivationMap);
public Map<String, Object> getTrendingMap() ;
public void setTrendingMap(Map<String, Object> trendingMap);
public Map<String, Object> getPredictionMap() ;
public void setPredictionMap(Map<String, Object> predictionMap);
public Map<String, Object> getDistinctionMap() ;
public void setDistinctionMap(Map<String, Object> distinctionMap);
}
------------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.estimation;
import java.util.Map;
import org.tinos.view.obj.WordFrequency;
public interface RatioMap {
                Map<String, EmotionSample> getEmotionSampleMap(Map<Integer, WordFrequency> wordFrequencyMap
                        , Map<String, Object> positive, Map<String, Object> negative);
        void getEmotionRatio(Map<String, EmotionSample> emotionSampleMap, double positiveCount
                        , double negativeCount, double medCount);
        void getMotivationRatio(Map<String, EmotionSample> emotionSampleMap, double sumOfEmotion);
        void getCorrelationRatio(Map<String, EmotionSample> emotionSampleMap, double sumOfEmotion);
        void getContinusRatio(Map<String, EmotionSample> emotionSampleMap, double emotionRatio);
        void getTrendsRatio( Map<String, EmotionSample> emotionSampleMap);
        void getPredictionRatio(Map<String, EmotionSample> emotionSampleMap);
        void getGuessRatio(Map<String, EmotionSample> emotionSampleMap);
        void getSensingRatio(Map<String, EmotionSample> emotionSampleMap);
        double findTotalPositiveCount(Map<String, EmotionSample> emotionSampleMap);
        double findTotalNegativeCount(Map<String, EmotionSample> emotionSampleMap);
        double findTotalKeyCount(Map<String, EmotionSample> emotionSampleMap);
        void getMotivation(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> motivation);
        void getTrending(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> trending);
        void getPrediction(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> prediction);
        void getDistinction(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> distinction);
        Map<String, EmotionSample> getEnvironmentSampleMap(Map<Integer, WordFrequency> wordFrequencyMap);
}
-----------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.estimation;
public class EmotionSample{
public double getPositiveCount() {
                return positiveCount;
        }
public void setPositiveCount(double positiveCount) {
                this.positiveCount= positiveCount;
        }
public double getNegativeCount() {
                return negativeCount;
        }
public void setNegativeCount(double negativeCount) {
                this.negativeCount= negativeCount;
        }public double getEmotionRatio() {
                return emotionRatio;
        }
public void setEmotionRatio(double emotionRatio) {
                this.emotionRatio= emotionRatio;
        }
public double getMotivationRatio() {
                return motivationRatio;
        }
public void setMotivationRatio(double motivationRatio) {
                this.motivationRatio= motivationRatio;
        }
public double getCorrelationRatio() {
                return correlationRatio;
        }
public void setCorrelationRatio(double correlationRatio) {
                this.correlationRatio= correlationRatio;
        }
public double getContinusRatio() {
                return continusRatio;
        }
public void setContinusRatio(double continusRatio) {
                this.continusRatio= continusRatio;
        }
public double getTrendsRatio() {
                return trendsRatio;
        }
public void setTrendsRatio(double trendsRatio) {
                this.trendsRatio= trendsRatio;
        }
public double getPredictionRatio() {
                return predictionRatio;
        }
public void setPredictionRatio(double predictionRatio) {
                this.predictionRatio= predictionRatio;
        }
public double getGuessRatio() {
                return guessRatio;
        }
public void setGuessRatio(double guessRatio) {
                this.guessRatio= guessRatio;
        }
public double getSensingRatio() {
                return sensingRatio;
        }
public void setSensingRatio(double sensingRatio) {
                this.sensingRatio= sensingRatio;
        }
public void setMedCount(int frequency) {
                this.medCount= frequency;
        }
```

```
public double getMedCount() {
                return medCount;
        }
public String getMotivation() {
                return motivation;
        }
public void setMotivation(String motivation) {
                this.motivation= motivation;
        }
public String getTrending() {
                return trending;
        }
public void setTrending(String trending) {
                this.trending= trending;
        }
public String getPrediction() {
                return prediction;
        }
public void setPrediction(String prediction) {
                this.prediction= prediction;
        }
public void setMedCount(double medCount) {
                this.medCount= medCount;
        }
public String getDistinction() {
                return distinction;
        }
public void setDistinction(String distinction) {
                this.distinction= distinction;
        }
        double positiveCount;
        double medCount;
        double negativeCount;
        double emotionRatio;
        double motivationRatio;
        double correlationRatio;
        double continusRatio;
        double trendsRatio;
        double predictionRatio;
        double guessRatio;
        double sensingRatio;
        String motivation;
        String trending;
        String prediction;
        String distinction;
}
package
org.tinos.emotion.estimation.imp;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import org.tinos.emotion.estimation.EmotionSample;
import org.tinos.emotion.estimation.RatioMap;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
public class RatioMapImp implements RatioMap{
        @Override
public Map<String, EmotionSample> getEmotionSampleMap(Map<Integer, WordFrequency> wordFrequencyMap, Map<String, Object>
positive, Map<String, Object> negative) {
        Map<String, EmotionSample> output= new HashMap<>();
                for(int i= wordFrequencyMap.size() - StableData.INT_ONE;
                i >= StableData.INT_ZERO;
                i--) {
                if(wordFrequencyMap.get(i).getWord().length() > StableData.INT_ONE) {

                EmotionSample emotionSample;
        if(output.containsKey(wordFrequencyMap.get(i).getWord())) {
                emotionSample= output.get(wordFrequencyMap.get(i).getWord());
                }else {
                emotionSample= new EmotionSample();
                }
        if(positive.containsKey(wordFrequencyMap.get(i).getWord())) {
                emotionSample.setPositiveCount(wordFrequencyMap.get(i).getFrequency());
                }else if(negative.containsKey(wordFrequencyMap.get(i).getWord())) {
                emotionSample.setNegativeCount(wordFrequencyMap.get(i).getFrequency());
                }else {
                emotionSample.setMedCount(wordFrequencyMap.get(i).getFrequency()}
                output.put(wordFrequencyMap.get(i).getWord(), emotionSample);
                        }}
                return output;
        }
        @Override
public void getMotivationRatio(Map<String, EmotionSample> emotionSampleMap, double sumOfEmotion) {
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= emotionSampleMap.get(word);
                        emotionSample.setMotivationRatio(emotionSample.getEmotionRatio()/sumOfEmotion);
                        emotionSampleMap.put(word, emotionSample);
                }}@Override
public void getCorrelationRatio(Map<String, EmotionSample> emotionSampleMap, double sumOfEmotion) {
                        Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                        while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                emotionSample.setCorrelationRatio((emotionSample.getPositiveCount()
+ emotionSample.getNegativeCount()+ emotionSample.getMedCount())/sumOfEmotion);
                emotionSampleMap.put(word, emotionSample);
                        }}
                @Override
public void getContinusRatio(Map<String, EmotionSample> emotionSampleMap, double emotionRatio) {
                        Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                        while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                emotionSample.setContinusRatio((emotionSample.getPositiveCount()
+ emotionSample.getNegativeCount()+ emotionSample.getMedCount()) * emotionRatio);
                emotionSampleMap.put(word, emotionSample);
                        }}
                @Override
public void getTrendsRatio(Map<String, EmotionSample> emotionSampleMap) {
                        Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                        while(Iterator.hasNext()) {

                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                emotionSample.setTrendsRatio(emotionSample.getEmotionRatio() * emotionSample.getContinusRatio()
                        * emotionSample.getCorrelationRatio());
                emotionSampleMap.put(word, emotionSample);
                        }}
                @Override
public void getPredictionRatio(Map<String, EmotionSample> emotionSampleMap) {
                        Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                        while(Iterator.hasNext()) {

                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                emotionSample.setPredictionRatio(emotionSample.getMotivationRatio()*emotionSample.getCorrelationRatio());
                emotionSampleMap.put(word, emotionSample);
                        }}@Override
public void getGuessRatio(Map<String, EmotionSample> emotionSampleMap) {
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                emotionSample.setGuessRatio(emotionSample.getPredictionRatio()*emotionSample.getTrendsRatio());
                emotionSampleMap.put(word, emotionSample);
                }}
                        @Override
public void getSensingRatio(Map<String, EmotionSample> emotionSampleMap) {
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
        if(0==emotionSample.getTrendsRatio()) {
                        emotionSample.setSensingRatio(0);
                }else {
                        emotionSample.setSensingRatio(emotionSample.getPredictionRatio()/emotionSample.getTrendsRatio());
                }
                emotionSampleMap.put(word, emotionSample);
                }}
                        @Override
public double findTotalPositiveCount(Map<String, EmotionSample> emotionSampleMap) {
                double output= StableData.INT_ONE;
                Iterator<String> Iterator=emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                output += emotionSample.getPositiveCount();
                }
                return output;
                        }
                        @Override
public double findTotalNegativeCount(Map<String, EmotionSample> emotionSampleMap) {
                double output= StableData.INT_ONE;
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                output += emotionSample.getNegativeCount();
                }
                return output;
                        } @Override
public void getEmotionRatio(Map<String, EmotionSample> emotionSampleMap, double positiveCount, double negativeCount, double
medCount) {

                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                double negRate= emotionSample.getNegativeCount()/negativeCount;
                double posRate= emotionSample.getPositiveCount()/positiveCount;
                double medRate= emotionSample.getMedCount()/medCount;
                emotionSample.setEmotionRatio(negRate+ posRate+ medRate);
                emotionSampleMap.put(word, emotionSample);
                }}
                        @Override
public double findTotalKeyCount(Map<String, EmotionSample> emotionSampleMap) {
                double output= StableData.INT_ONE;
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                String word= Iterator.next();
                EmotionSample emotionSample= emotionSampleMap.get(word);
                output += emotionSample.getNegativeCount()+ emotionSample.getPositiveCount()
+ emotionSample.getMedCount();
                }
                return output;
                        }
                        @Override
public void getMotivation(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> motivation) {
                Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
```

```
			while(Iterator.hasNext()) {
			String word= Iterator.next();
			EmotionSample emotionSample= emotionSampleMap.get(word);
		if(motivation.containsKey(word)) {
				emotionSample.setMotivation(motivation.get(word).toString());
			}
			emotionSampleMap.put(word, emotionSample);
			}}
				@Override
public void getTrending(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> trending) {
			Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
			while(Iterator.hasNext()) {
			String word= Iterator.next();
			EmotionSample emotionSample= emotionSampleMap.get(word);
		if(trending.containsKey(word)) {
				emotionSample.setTrending(trending.get(word).toString());
			}
			emotionSampleMap.put(word, emotionSample);
			}}
				@Override
public void getPrediction(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> prediction) {
			Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
			while(Iterator.hasNext()) {
			String word= Iterator.next();
			EmotionSample emotionSample= emotionSampleMap.get(word);
		if(prediction.containsKey(emotionSample.getTrending())) {
				emotionSample.setPrediction(prediction.get(emotionSample.getTrending()).toString());
			} else if(prediction.containsKey(emotionSample.getMotivation())) {
				emotionSample.setPrediction(prediction.get(emotionSample.getMotivation()).toString());
			}
			emotionSampleMap.put(word, emotionSample);
			}}
				@Override
public void getDistinction(Map<String, EmotionSample> emotionSampleMap, Map<String, Object> distinction) {
			Iterator<String> Iterator= emotionSampleMap.keySet().iterator();
			while(Iterator.hasNext()) {
			String word= Iterator.next();
			EmotionSample emotionSample= emotionSampleMap.get(word);
		if(distinction.containsKey(word)) {
				emotionSample.setDistinction(distinction.get(word).toString());
			}
			emotionSampleMap.put(word, emotionSample);
			}}
				@Override
public Map<String, EmotionSample> getEnvironmentSampleMap(Map<Integer, WordFrequency> wordFrequencyMap) {
		Map<String, EmotionSample> output= new HashMap<>();
			for (int i= wordFrequencyMap.size() - StableData.INT_ONE;i >= StableData.INT_ZERO;i--) {
		if(wordFrequencyMap.get(i).getWord().length() > StableData.INT_ONE) {
				EmotionSample emotionSample= new
EmotionSample();if(!output.containsKey(wordFrequencyMap.get(i).getWord())) {
output.put(wordFrequencyMap.get(i).getWord(), emotionSample);
				}}}
			return output;
				}}
-----------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.engine;
import java.io.IOException;
import java.util.Map;
import org.tinos.emotion.estimation.EmotionSample;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.sensing.ortho.fhmm.SensingMap;
import org.tinos.sensing.ortho.fhmm.imp.SensingMapImp;
public class LenovoInit{
public SensingMap getSensingMap() {
			return sensingMap;
			}
public void setSensingMap(SensingMap sensingMap) {
```

```
                    this.sensingMap= sensingMap;
                    }
public EnvironmentInit getEnvironmentInit() {
                    return environmentInit;
                    }
public void setEnvironmentInit(EnvironmentInit environmentInit) {
                    this.environmentInit= environmentInit;
                    }
          private SensingMap sensingMap;
          private EnvironmentInit environmentInit;
                    @SuppressWarnings("unused")
public static void main(String[] argv) throws IOException {
                    String text= "  //20210727 采样数据仅保存一次，之后相同全部删除。";
LenovoInit lenovoInit= new LenovoInit();
                    lenovoInit.init(text);
          Map<String, EmotionSample> environmentSampleMap= lenovoInit.getEnvironmentInit().getEmotionSampleMap();
          Map<String, Object> lenovo= lenovoInit.getSensingMap().getLenovoMap();
                    }
public void init(String text) throws IOException {
                    environmentInit= new EnvironmentInit();
                    environmentInit.init(text);
                    sensingMap= new SensingMapImp();
                    sensingMap.initLenovoMap(environmentInit.getAnalyzer());
                    }
public void initExcludeAnalyzer(String text, Analyzer analyzer) throws IOException {
                    environmentInit= new EnvironmentInit();
                    environmentInit.initExcludeAnalyzer(text, analyzer);
                    sensingMap= new SensingMapImp();
                    sensingMap.initLenovoMap(environmentInit.getAnalyzer());
                    }}
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.engine;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import org.tinos.emotion.estimation.EmotionSample;
import org.tinos.emotion.estimation.RatioMap;
import org.tinos.emotion.estimation.imp.RatioMapImp;
import org.tinos.emotion.ortho.fhmm.EmotionMap;
import org.tinos.emotion.ortho.fhmm.imp.EmotionMapImp;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.view.obj.WordFrequency;
public class EnvironmentInit{
public EmotionMap getEmotionMap() {
                    return emotionMap;
                    }
public void setEmotionMap(EmotionMap emotionMap) {
                    this.emotionMap= emotionMap;
                    }
public Analyzer getAnalyzer() {
                    return analyzer;
                    }
public void setAnalyzer(Analyzer analyzer) {
                    this.analyzer= analyzer;
                    }public Map<String, Object> getPositive() {
                    return positive;
                    }
public void setPositive(Map<String, Object> positive) {
                    this.positive= positive;
                    }
public Map<String, Object> getNegative() {
                    return negative;
                    }
public void setNegative(Map<String, Object> negative) {
                    this.negative= negative;
                    }
```

```
public Map<String, Object> getMotivation() {
                    return motivation;
                    }
public void setMotivation(Map<String, Object> motivation) {
                    this.motivation= motivation;
                    }
public Map<String, Object> getTrending() {
                    return trending;
                    }
public void setTrending(Map<String, Object> trending) {
                    this.trending= trending;
                    }
public Map<String, Object> getPrediction() {
                    return prediction;
                    }
public void setPrediction(Map<String, Object> prediction) {
                    this.prediction= prediction;
                    }
public List<String> getSets() {
                    return sets;
                    }
public void setSets(List<String> sets) {
                    this.sets= sets;
                    }
public RatioMap getRationMap() {
                    return rationMap;
                    }
public void setRationMap(RatioMap rationMap) {
                    this.rationMap= rationMap;
                    }
public Map<Integer, WordFrequency> getWordFrequencyMap() {
                    return wordFrequencyMap;
                    }
public void setWordFrequencyMap(Map<Integer, WordFrequency> wordFrequencyMap) {
                    this.wordFrequencyMap= wordFrequencyMap;
                    }
public Map<String, EmotionSample> getEmotionSampleMap() {
                    return emotionSampleMap;
                    }
public void setEmotionSampleMap(Map<String, EmotionSample> emotionSampleMap) {
                    this.emotionSampleMap= emotionSampleMap;
                    }
public double getPositiveCount() {
                    return positiveCount;
                    }
public void setPositiveCount(double positiveCount) {
                    this.positiveCount= positiveCount;
                    }
public double getNegativeCount() {
                    return negativeCount;
                    }
public void setNegativeCount(double negativeCount) {
                    this.negativeCount= negativeCount;
                    }
public double getTotalCount() {
                    return totalCount;
                    }
public void setTotalCount(double totalCount) {
                    this.totalCount= totalCount;
                    }
public Map<String, Object> getDistinction() {
                    return distinction;
                    }
public void setDistinction(Map<String, Object> distinction) {
                    this.distinction= distinction;
                    }
          private EmotionMap emotionMap;
```

```
        private Analyzer analyzer;
        private Map<String, Object> positive;
        private Map<String, Object> negative;
        private Map<String, Object> motivation;
        private Map<String, Object> trending;
        private Map<String, Object> prediction;
        private Map<String, Object> distinction;
        private List<String> sets;
        private RatioMap rationMap;
        private Map<Integer, WordFrequency> wordFrequencyMap;
        private Map<String, EmotionSample> emotionSampleMap;
        private double positiveCount;
        private double negativeCount;
        private double totalCount;
                @SuppressWarnings("unused")
public static void main(String[] argv) throws IOException {
                String text= "  //20210727 采样数据仅保存一次，之后相同全部删除。";
EnvironmentInit environmentInit= new EnvironmentInit();
                environmentInit.init(text);
        Map<String, EmotionSample> environmentSampleMap= environmentInit.getEmotionSampleMap();
                }
public void init(String text) throws IOException {
                emotionMap= new EmotionMapImp();
                emotionMap.initMotivationMap();
                emotionMap.initTrendingMap();
                emotionMap.initPredictionMap();
                emotionMap.initDistinctionMap();
                analyzer= new CogsBinaryForestAnalyzerImp();
                analyzer.init();
                motivation= emotionMap.getMotivationMap();
                trending= emotionMap.getTrendingMap();
                prediction= emotionMap.getPredictionMap();
                distinction= emotionMap.getDistinctionMap();
                sets= analyzer.parserString(text);
                wordFrequencyMap= analyzer.getWordFrequencyByReturnSortMap(sets);
                rationMap= new RatioMapImp();
                emotionSampleMap= rationMap.getEnvironmentSampleMap(wordFrequencyMap);
                rationMap.getMotivation(emotionSampleMap, motivation);
                rationMap.getTrending(emotionSampleMap, trending);
                rationMap.getPrediction(emotionSampleMap, prediction);
                rationMap.getDistinction(emotionSampleMap, distinction);
                }
public void initExcludeAnalyzer(String text, Analyzer analyzerInput) throws IOException {
                emotionMap= new EmotionMapImp();
                emotionMap.initMotivationMap();
                emotionMap.initTrendingMap();
                emotionMap.initPredictionMap();
                emotionMap.initDistinctionMap();
                analyzer= analyzerInput;
                motivation= emotionMap.getMotivationMap();
                trending= emotionMap.getTrendingMap();
                prediction= emotionMap.getPredictionMap();
                distinction= emotionMap.getDistinctionMap();
                sets= analyzerInput.parserString(text);
                wordFrequencyMap= analyzerInput.getWordFrequencyByReturnSortMap(sets);
                rationMap= new RatioMapImp();
                emotionSampleMap= rationMap.getEnvironmentSampleMap(wordFrequencyMap);
                rationMap.getMotivation(emotionSampleMap, motivation);
                rationMap.getTrending(emotionSampleMap, trending);
                rationMap.getPrediction(emotionSampleMap, prediction);
                rationMap.getDistinction(emotionSampleMap, distinction);
                }
public void initFromEmotion(Map<Integer, WordFrequency> getWordFrequencyMap) throws IOException {
                emotionMap= new EmotionMapImp();
                emotionMap.initMotivationMap();
                emotionMap.initTrendingMap();
                emotionMap.initPredictionMap();
```

```
                    emotionMap.initDistinctionMap();
                    //parser sentence
                    motivation= emotionMap.getMotivationMap();
                    trending= emotionMap.getTrendingMap();
                    prediction= emotionMap.getPredictionMap();
                    distinction= emotionMap.getDistinctionMap();
                    //map rationMap= new RatioMapImp();
                    emotionSampleMap= rationMap.getEnvironmentSampleMap(getWordFrequencyMap);
                    rationMap.getMotivation(emotionSampleMap, motivation);
                    rationMap.getTrending(emotionSampleMap, trending);
                    rationMap.getPrediction(emotionSampleMap, prediction);
                    rationMap.getDistinction(emotionSampleMap, distinction);
                    }
public void initFromEmotionExcludeEmotion(Map<Integer, WordFrequency> getWordFrequencyMap, EmotionMap emotionMapInput)
throws IOException {
                    emotionMap= emotionMapInput;
                    motivation= emotionMap.getMotivationMap();
                    trending= emotionMap.getTrendingMap();
                    prediction= emotionMap.getPredictionMap();
                    distinction= emotionMap.getDistinctionMap();
                    rationMap= new RatioMapImp();
                    emotionSampleMap= rationMap.getEnvironmentSampleMap(getWordFrequencyMap);
                    rationMap.getMotivation(emotionSampleMap, motivation);
                    rationMap.getTrending(emotionSampleMap, trending);
                    rationMap.getPrediction(emotionSampleMap, prediction);
                    rationMap.getDistinction(emotionSampleMap, distinction);
                    }}
------------------------------------------------------------------------------------------------------------------------------
package org.tinos.emotion.engine;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import org.tinos.emotion.estimation.EmotionSample;
import org.tinos.emotion.estimation.RatioMap;
import org.tinos.emotion.estimation.imp.RatioMapImp;
import org.tinos.emotion.ortho.fhmm.EmotionMap;
import org.tinos.emotion.ortho.fhmm.imp.EmotionMapImp;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.view.obj.WordFrequency;
public class EmotionInit{
public EmotionMap getEmotionMap() {
                    return emotionMap;
                    }
public void setEmotionMap(EmotionMap emotionMap) {
                    this.emotionMap= emotionMap;
                    }public Analyzer getAnalyzer() {
                    return analyzer;
                    }
public void setAnalyzer(Analyzer analyzer) {
                    this.analyzer= analyzer;
                    }
public Map<String, Object> getPositive() {
                    return positive;
                    }
public void setPositive(Map<String, Object> positive) {
                    this.positive= positive;
                    }
public Map<String, Object> getNegative() {
                    return negative;
                    }
public void setNegative(Map<String, Object> negative) {
                    this.negative= negative;
                    }
public Map<String, Object> getMotivation() {
                    return motivation;
                    }
```

```
public void setMotivation(Map<String, Object> motivation) {
			this.motivation= motivation;
			}
public Map<String, Object> getTrending() {
			return trending;
			}
public void setTrending(Map<String, Object> trending) {
			this.trending= trending;
			}
public Map<String, Object> getPrediction() {
			return prediction;
			}
public void setPrediction(Map<String, Object> prediction) {
			this.prediction= prediction;
			}public List<String> getSets() {
			return sets;
			}
public void setSets(List<String> sets) {
			this.sets= sets;
			}
public RatioMap getRationMap() {
			return rationMap;
			}
public void setRationMap(RatioMap rationMap) {
			this.rationMap= rationMap;
			}
public Map<Integer, WordFrequency> getWordFrequencyMap() {
			return wordFrequencyMap;
			}
public void setWordFrequencyMap(Map<Integer, WordFrequency> wordFrequencyMap) {
			this.wordFrequencyMap= wordFrequencyMap;
			}
public Map<String, EmotionSample> getEmotionSampleMap() {
			return emotionSampleMap;
			}
public void setEmotionSampleMap(Map<String, EmotionSample> emotionSampleMap) {
			this.emotionSampleMap= emotionSampleMap;
			}
public double getPositiveCount() {
			return positiveCount;
			}
public void setPositiveCount(double positiveCount) {
			this.positiveCount= positiveCount;
			}
public double getNegativeCount() {
			return negativeCount;
			}
public void setNegativeCount(double negativeCount) {
			this.negativeCount= negativeCount;
			}
public double getTotalCount() {
			return totalCount;
			}
public void setTotalCount(double totalCount) {
			this.totalCount= totalCount;
			}
		private EmotionMap emotionMap;
		private Analyzer analyzer;
		private Map<String, Object> positive;
		private Map<String, Object> negative;
		private Map<String, Object> motivation;
		private Map<String, Object> trending;
		private Map<String, Object> prediction;
		private List<String> sets;
		private RatioMap rationMap;
		private Map<Integer, WordFrequency> wordFrequencyMap;
		private Map<String, EmotionSample> emotionSampleMap;
```

```
            private double positiveCount;
            private double negativeCount;
            private double totalCount;
public static void main(String[] argv) throws IOException {
                        String text= "  //20210727 采样数据仅保存一次，之后相同全部删除。";
EmotionInit emotionInit= new EmotionInit();
                        emotionInit.init(text);
                        }
public void init(String text) throws IOException {
                        emotionMap= new EmotionMapImp();
                        emotionMap.initNegativeMap();
                        emotionMap.initPositiveMap();
                        analyzer= new CogsBinaryForestAnalyzerImp();
                        analyzer.init();
                        positive= emotionMap.getPositiveMap();
                        negative= emotionMap.getNegativeMap();
                        sets= analyzer.parserString(text);
                        wordFrequencyMap= analyzer.getWordFrequencyByReturnSortMap(sets);
                        rationMap= new RatioMapImp();
                        emotionSampleMap= rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);
                        positiveCount= rationMap.findTotalPositiveCount(emotionSampleMap);
                        negativeCount= rationMap.findTotalNegativeCount(emotionSampleMap);
                        totalCount= rationMap.findTotalKeyCount(emotionSampleMap);
                        }
public void initExcludeAnalyzer(String text, Analyzer analyzerInput, EmotionMap emotionMapInput) throws IOException {
                        emotionMap= emotionMapInput;
                        analyzer= analyzerInput;
                        positive= emotionMap.getPositiveMap();
                        negative= emotionMap.getNegativeMap();
                        sets= analyzer.parserString(text);
                        wordFrequencyMap= analyzer.getWordFrequencyByReturnSortMap(sets);
                        rationMap= new RatioMapImp();
                        emotionSampleMap= rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);
                        positiveCount= rationMap.findTotalPositiveCount(emotionSampleMap);
                        negativeCount= rationMap.findTotalNegativeCount(emotionSampleMap);
                        totalCount= rationMap.findTotalKeyCount(emotionSampleMap);
                        }}
-----------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.test;
import java.io.IOException;
import org.tinos.behavior.ICA.InitBehaviorICAKernel;
import matrixProcessor.CnnMeasure;
public class SuccessICATest{
public double[][] kernelCNN;
public void getKernelCNN(double[][] kernel) {
                        kernelCNN= new CnnMeasure().getCnnMeansure(kernel);
                        }
public static void main(String[] argv) throws IOException {
                        String text1= "关于成瘾性的戒除方式，上瘾在医学上普遍定义为一种具有精神依赖并长期导致健康危害性的行为。
\r\n"
+
"关于成瘾的溯源有很多因素，其中最重要的是依赖。因为长期的依赖导致自身某种缺陷逐渐丧失而\r\n" +
"对成瘾物体产生不可替代性。通过这个推论，可以初步来定义戒断瘾欲，最有效的方式是替代和引导。\r\n"+ "替代物，本身也是有强
烈制瘾性，和危害性，但是危害要小于原物。通过替代和强制减少剂量和精洗\r\n" +
"脑教育，通过一个时间周期达到戒除。中间有戒断反应，需要观察。引导，是在对没有成瘾并属于易\r\n"+ "感染群体进行教育和传授
方式，提高群体的免疫力和排斥力。上瘾不是欲望。欲望是生物的应激性进\r\n"+ "化的产物，是与生俱来的。上瘾是一种外力干涉造
成的依赖。上瘾的级别有很多种。医学有相关严谨\r\n"+ "的打分段，其中毒瘾大于烟瘾大于网瘾。最有效的戒除手段就是环境和生活
方式的选择。很多时候\r\n" +
"环境不是很美好，生活方式充满了隐患，人的精神会产生误差，这个时候受体为不稳定态，极易接触\r\n"+ "成瘾源。当环境无法改变
的时候，我们需要改变自己，选择一个愉悦的生活方式，进行自我心里疏导，\r\n" +
"很容易排斥上瘾源。其中这些词汇是非常有价值的精神药物：自信，豁达，友善，分享 等等。\r\n" +
"一些成瘾的受体，普遍有某种倾向：奢靡，闭塞，强迫，空虚 等等。这里不是贬义，只是因为长期的环境\r\n"
+
"因素不是那么美好导致了一些思维误差。所以引导是非常重要的。改变人的不是能力，而是选择和环境。\r\n"
+
```

"如果环境不是很完美，那么选择一个健康的生活方式，是非常重要的。";

String text2= "我们在这个三维世界能听到许多答案，却无法解释它，最后物理学用 T 来表示。带着这个疑问，我开始

寻求答案来解释。语文和数学对时间的描述基于我罗瑶光的归纳为是事物发展过程中的某一点基于经典数学的观测参照。我开始"

+ "深思，既然是参照，那么必定是有参照物和观测物 2 种模式。我又深思，既然是 2 种模式改变，必定引起

时间的不准确性，那么我定义为时间不是经典物理学中的概念。而是量子物理的一个分支。我又深思，如果时间因为参照物和

观测物的不同"

+ "，那么这个时间肯定是有变化规律的，这个规律肯定是一个抽象函数，我定义为 T(x)怎么求解时函数？ 我深思了 10 年。或许我

罗瑶光是世界第一个定义时函数的人，我可能成为不了第一"

+ "个能够利用时函数穿梭高维空间的人。霍金死了，傅里叶死了，爱因斯坦也死了，薛定谔死了，海森堡

死了，狄拉克死了，他们也许发现了时函数的一些规律。我又深思了许久。时函数的解析一直困扰着我。怎么求解？这些先贤

给我巨大的视野。\r\n" +

"1 时间是一个事物发展的过程，而事物发展，可以用 p(t)表示，狄拉克说事物的发展通常用向量集合表示，我归纳为 |p(t)> 在这里

感谢狄拉克先生。\r\n" +

"2 因为观测物不一样和观测角度不一样，那么这个 事物发展的向量集肯定也会被扭曲，那么我用量子力学的 <m(t)|p(t)> 来表示 观

测点与事物运动点的内积的狄拉克本征量表示。\r\n" +

"3 以为 2 是个非常复杂的逻辑，我采用理想的正交表达，<m(t)|p(t)> 其实是一个理想函数。\r\n" +

"4.1 怎么解析这个狄拉克方程我又深思的许久，我想到以前我写的狄拉克 傅里叶和薛定谔是好朋友，于是我

用薛定谔含时函数来解析，为什么用含时，因为它同时有时间和运动轨迹的观测点。于是我得出(1-iht)|m(0)> 与 (1-iht"

+ ")|p(0)> 正交这个 t 无法消除，看来我方法也许是错的。。。\r\n"+ "4.2 我想用傅里叶咋样？ 于是我得出\r\n" +

"dp/sq(2*pi) * pof(m)(p) e**ipt 与 dp/sq(2*pi) * pof (p) e**ipt 正交，结合 4.1 和 4.2 我得到 2 个方程组\r\n"+ "1:(1-iht)|m(0)>=dp/sq(2*pi)

* pof(m)(p) e**ipt \r\n" +

"2:(1-iht)|p(0)>=dp/sq(2*pi) * pof (p)(p) e**ipt \r\n" +

"5,1-2=> (1-iht)(|m(0)>-|p(0)>)=dp/sq(2*pi) (pof(m)(p) e**ipt - pof (p)(p) e**ipt)\r\n"+ "6 事物运动最开始一定是静止的，我可得到

|m(0)>-|p(0)>=0\r\n" +

"7：dp/sq(2*pi) (pof(m)(p) e**ipt - pof (p)(p) e**ipt)=0\r\n" +

"8：我又想到了海森堡和斯塔函数 把傅里叶的 dp/2pi*eip(x1-x0) 用 deta 表示。。。\r\n"+ "-》intel（dx）*deta (t1-t0) m(t) - intel（dx）*deta

(t1-t0) p(t)=0 \r\n" +

"9: intel（dx）*deta (t1-t0) (m(t) -p(t))=0\r\n"+ "想到这里我发现我升华了。。。。\r\n" +

"deta (t1-t0)*m(t）=deta1(t1-t0)*p(t)。。。。\r\n"+ "我得到几个答案：\r\n" +

"1 事物运动 等于 观测运动，\r\n"+ "2 观测停止 等于 事物停止。\r\n" +

"3 观测轨迹 等于 事物轨迹。而这个轨迹就是一个傅里叶的 deta 波函数\r\n"+ "4 核武器由磁暴激发。\r\n" +

"5 纠缠态证明 平行宇宙存在。\r\n" +

"6 deta 的概率集确定光纤通信基础。\r\n" +

"或许霍金真的无法接受这个答案，宇宙没有开始，也没有结束，而万物始于一个 deta 弦，我又深思了许久，

既然宇宙没有过去未来，那说明了什么？\r\n" +

"1 说明了我们的认知被时函数误导了，因为我们这个世界所有的经典力学都是加入了时间变量进行思考和运

算的, 这是三维世界智慧体的禁区。\r\n"+ "2 我不敢再写下去，因为这些思维完全违背了这个世界的经典定律。。。

\r\n"+ "如果我罗瑶光是世界以第一个定义时函数的人，那我今天 初步定义为:时函数代表物体的运动轨迹和观测者

轨迹的量子函数关系式，是宇宙的维度的核心组成部分。这些年，我"

+ "总想成为一名程序员。很多年前，一位很有成就的教授跟我说程序员只是一个工人，如果我把才华定义为一个工人，那么我对不"

+ "起我这 20 多年来的,理论研究功底，因为很多程序员职位通过速成就能胜任。后来当我遇到一位顶级的科

研工程师同事，我发现，他的编程能力我真不敢恭维，可是他确是首席。。带着这份不羁，我慢慢才知道。他比我强

在细节和专

注。这个出发点是对的，\r\n" +

"deta (t1-t0)*m(t）=deta1(t1-t0)*p(t)\r\n" +

"很明确，我得到了这个初级公式。这个公式我深思，时函数肯定和 斯塔函数，傅里叶函数，观测者，运动者 这 4 个 因子有关,而且

观测者和运动者成正比，我得到一个理论答案：时间"

+ "可以任意膨胀和坍塌。显然，膨胀和坍塌的函数我无法从这个公式得到推理，我进行了很多年的思考。可是

我得到了另外一个答案：时间任意膨胀，必定无穷大，时间坍塌，必定无穷小。我又得到一个答案：时函数是一个

矢量函数。

\r\n" +

"1: deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t) =1 \r\n" +

"2: t= |m(x)> / |p(x)> 关于 2 是怎么来的，我是这样思考的，如果把经典物理学中 s=vt, 那么我把 s 当做观测者，把 v 当成发展事物，

那么有\r\n" +

"|m(x)>= |p(x)> * t\r\n" +

"这是求解狄拉克的熵，我很可惜。因为狄拉克在实验求解的时候，年老病逝。可是我没有停下，我又得到 2

个关系式：\r\n" +

"t= dp/sq(2*pi) * pof(m)(p) e**ipt / dp/sq(2*pi) * pof (p) e**ipt \r\n"+ "t= deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t)\r\n" +

"当我得到这个时候，我发现了真正恐怖。。如果在粒子在磁暴激发的时候，能够用时间膨胀函数抑制，那么这

个粒子就具有巨大的能量。而这个"
+ "能量我称之为时间函数膨胀势能。狄拉克的相对论说明有时间函数膨胀势能 必有 时间函数坍塌势能，如果用 tero(x) 和 tcol(x)表示则有 |tero(x)>+|tcol(x)>=1\r\n" +
"这就是狄拉克晚年论述为什么有反粒子的存在。可是我现在疑惑了；\r\n"+ "1：|tero(x)>+|tcol(x)>=1\r\n" +
"2：deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t) =1 \r\n" +
"=》|tero(x)>+|tcol(x)>= deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t) \r\n" +
"时间膨胀和时间坍塌也是由 观测者和运动者 决定的。而且是一个有归一性的矢量函数控制。用我们三维世界的经典思维解释 我得到这样一个"
+ "答案宇宙没有过去，没有未来，充满物质，却又虚无，而一切都是一个 deta 弦。这明显就是一个错误，却又存在，我又一次发现我的经"
+ "典思维再一次误导了我。。。。我的直觉告诉我我被耍了，我的出发点一开始是错误的，我不能用三维世界的经典定理来推导时函数。很悲"
+ "伤，我的出发点错误的，同时，很高兴，我通过错误的伪命题得到一个真确答案：|tero(x)>+|tcol(x)>= deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t) \r\n" +
"这公式有什么用？我思考了许久。\r\n" +
"我发现我又升华了。。。如果粒子辐射角向上激发能量释放，我们能创造一个对称的共轭坍塌，那么就能中和能量。什么意思？好比原子弹"
+ "爆炸了，却破坏力等于 0.。。。。而这个汉密尔顿反粒子（量子数学叫轭米粒子，量子力学叫汉密尔顿共轭）怎么实现？我又思考了很久"
+ "我又得到了答案。在平行纠缠态的异域 通过量子纠缠控制反应区量子集合，通过量子纠缠可以控制原子反
应。而这一切离不开时函数。而公式就是 \r\n"+ "|tero(x)>+|tcol(x)>= deta (t1-t0)*m(t）/ deta1(t1-t0)*p(t) \r\n" +
"推导了这个公式后，最近有看了一下，又稍微变化了一下：假设\r\n"+ "1 deta (t1-t0)/deta1(t1-t0)= 时间流逝比重\r\n" +
"2 |tero(x)>+|tcol(x)>= 中和时间能\r\n"+ "3 m(t）= 观测\r\n"+ "4 p(t)= 发展\r\n" +
"我得到一个公式：\r\n" +
"5 中和时间能= 时间流逝比重 * 观测 / 发展\r\n"+ "6 时间流逝比重= 中和时间能 * 发展 / 观测\r\n" +
"今天又看了下这个公式 6，我得到一个论证。在一种固定的观测态中，发展速度越快时间流逝比重大。以后星际穿梭 飞船飞行速度越快，时间"
+ "流逝比重越大，"
+ "意味着飞行过程时间应该指数级别比例缩短。几百光年的距离，未必要那么久的时间穿梭。不知道假设是不是正确的。我在思考这种固定的观测"
+ "态怎么模拟出来。我得到了一个推论结果，当物体超高速运动的时候，物体具有巨大的动能，这种能量能够和时间能进行公式转化，也可以转化"
+ "成力学，强大的力可以扭曲时空，达到瞬间转移。若果推论成立，物体在时空中带强力强能急速飞行，物体四周的时空会产生巨大引力扭曲环层，"
+ "罗氏猜想这个环层时空扭曲力能牵引物体，我定义为局部时空扭曲引力，这是瞬间转移的关键。";
String text3= "最新的知识工程结构中，传统的专家系统占据着主导的地位，可是世界的需求体系处在一个多变的运行环境，所以数据持久化理论是一个为之奋斗的目标。"
+ "人工智能软件也一样，逃避不了自然的更新所带来的种种弊端。人工智能何去何从，自然会规划它，正如达尔文的生物进化论一样，新的智能体系标准都是被需求自然选择出来，这就是我要表达的中心思想。\r\n" +
"过去 50 年里，一些经典的软件逃不过需求的抉择，最终枯黄暗淡，当然一些企业将产品拼命的重写升级，因为核心开发者的年龄老化，新的改造者无法掌握原始开发思想和理论，最后产品的品质遭受巨大的冲击，"
+ "损失惨重，一种新的软件开发理论需要被人所证实，这也就是我的思想。软件也一样，需要有自我的人工选择的进化体系。\r\n" +
"通过最近的 UNICORN AI\r\n" +
"软件的构造，设计和编码测试中，我发现了许多因空想而创造的计算机理论在实际的编程分析中有巨大的差异，我用的是 JAVA 为主的语言，我就发现 JAVA 的继承没有达到具有进化思想的语言标准，"
+ "但是 JAVA 在这个初期的进化标准测试中其方法论远远胜出 C/C++ ,我用 C 风格写 JAVA 程序并没有给我的实际编程带来种种麻烦，但是 JAVA 仍然需要改进，比如你抽象了一个父类，而你的子类的变"
+ "量函数还是需要在“OBJECT 父类=(子类)父类” 这样的写法中的才能做出子类运算。如果孙类又继承子类，怎么让 OBJECT 得到孙类？（我的用的是 OBJECT 子类继承父类，然后 OBJECT "
+ "子类=(孙类)子类 。这样孙类得到了运算），可是这就是一个动态内存结构分配的大问题！设计的相当繁琐。JAVA 还停留在初级语言进化级别，没有具备高级的进化思想。其次，子类如果有多个孙类，"
+ "也只有子类可以运算，父类就被无法作出相应的运算。这也是一个诟病，难道再加上 OBJECT 子类=(孙类) 子类，OBJECT 父类=(子类)父类 来实现？这就更加繁琐了。\r\n" +
"通过上面的描述，我有自己的看法，可是我还是选择了 JAVA，即使繁琐，但是没有任何错误，因为用底层语言来实现就会更加繁琐。陷阱更多。\r\n" +
"人工智能选择了 JAVA 是一个自然的抉择。JAVA 和 C#都是高层语言，可是 JAVA 的个性就是天生对数据来处理的，因为 JAVA 早期是一个 WEB 语言，WEB 处理数据信息有独特的优势，这是 JAVA 进化"
+ "为数据分析语言的一个真实的例子。C#在这个问题里一直在改进自己，类似 JAVA 一样，甚至和 JAVA 一样，可是没有一个体系来评估它。早期应用 JAVA 的 WEB 数据工程师也不会转移到 C#.所以 C#的最大优势还是仅仅在WINDOWS 上

```
			的控件应用。\r\n" +
"通过这段的描述，仅仅证明任何一种语言的最大优势也仅仅体现在它诞生之初的创造理论和思想。所以 JAVA
			和 C#根本就没有什么可比性。因为他们最原始的创造理论，体系和思想结构就不一样。如果真的"
	+ "JAVA 和 C#不倒，最后，通过进化的思想预测，JAVA 最后走图形，大数据分析，WEB,方向，而 C# 应该
			走界面，控件，WINDOWS 设备集成方向。\r\n" +
"人工智能软件的进化主要分为父类的更新，子类的变异和继承。现在的许多人工智能软件因为需求关系的制
			约，导致创造思想的缺乏，父类被写死了，无法得到应有的适应扩展，比如 ORACLE 的数据库 ETL"+ "，仅仅在处理数据仓库领域有巨大价值，无法扩展到数据可视化，并行运算等领域。德国的 KNIME 也是因
			为父类的写死，结果插件很多 API 都不支持，实例证明，我用 SWT 写插件界面，就无法实现。我"
	+ "在节点里面导入自己的数据库 API,它就要我在软件的配置选项里面去导入，这就是父类写死的诟病。\r\n"+ "当然有很多细节的问题，ORACLE ETL 和 KNIME DM 都不失为成功大作。上面提到的是父类写死没有得到进化论的思想。然后评论一下子类变异。\r\n" +
"JAVA 处理子类函数是比较完美的，用过 JAVA 开发大型项目的人都相当有经验处理接口和继承。可是 JAVA
			有没有变异的特性呢？可以说无，比如我举个例子，当父类 PUBLIC 属性 1=0;，子类就无"
+ "法在 PUBLIC 属性 1=1 了，这就是一个变异失效的问题。JAVA 很灵活，但是不够脚本语言灵活。其次我要说的是 JAVA 的变异是带引号的变异，其特点就是子类修改父类函数，JAVA 的子类是可以"
+ "修改父类的同名函数处理过程的。不过你要让子类和父类的函数名一样，这是一个 JAVA 默认的机制，先执行父类同名，再执行子类同名。然后返回到父类，然后返回的过程。所以同名函数可以在子类里得"
+ "到修改，保证了参数变异。这样，软件在实际的编写过程中也非常的灵活和独到。\r\n" +
"最后通过上述的语言进化思想，程序进化思想的表述，我有一个很深的体会。每一种语言要根深蒂固，需要有它的需求，它的功能在需求中要有选择的得到进化。不然，这就是语言被淘汰的最大原因，我不喜欢看"
+ "到当今世界上各种语言层出不穷，这就是许多语言没有得到进化，体现不了需求的最大诟病。其次，语言需要扩展，高级语言的 API 类库和一些架构体系的出现是一个很好的扩展证明。最后是变异，类似脚"
+ "本语言，灵活，方便。\r\n" +
"那么软件呢？软件也一样，选择一门适应自己需求的语言来设计尤为重要。其次，软件的架构要有松耦度，类似于 OSGI,FELIX 那样,进行组件持久化，KNIME 的 OSGI 思想和 LIFERAY 的 OSGI 思想是"
+ "一致的，虽然 API  设计风格不一样，但是效果都很笃厚。\r\n" +
"生物需要有达尔文思想，人工智能同样也存在，这是需求持久化的基础。这也是我研发 UNICORN AI 平台的基本条件。";
			//ICA kernel
			double[][] kernel= new double[3][];
			kernel[0]= new InitBehaviorICAKernel().getBehaviorICAKernel(text1);
			kernel[1]= new InitBehaviorICAKernel().getBehaviorICAKernel(text2);
			kernel[2]= new InitBehaviorICAKernel().getBehaviorICAKernel(text3);
			SuccessICATest successICATest=new SuccessICATest();
			successICATest.getKernelCNN(kernel);
			for(int i=0;
			i<successICATest.kernelCNN.length;
			i++) {
				for(int j=0;
			j<successICATest.kernelCNN[0].length;
			j++) {
			System.out.print(successICATest.kernelCNN[i][j]+ " ");
				}
				System.out.println();
			}
			//do ICA normalization
			//do ROBUST ICA
			//do map
			//do reduce sets
			//sets Turing
			}}
------------------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.test;
import java.io.IOException;
import org.tinos.behavior.ICA.InitBehaviorICAKernel;
import org.tinos.view.stable.StableData;
public class LiterarinessLevelTest{
			double[] literarinessLevel;
			double literarinessDuration;
public void getEducationLevel(double[][] measurement) {
			literarinessLevel= new double[measurement.length];
			for(int i= StableData.INT_ZERO;i< measurement.length;i++) {
				literarinessLevel[i]=(measurement[i][StableData.INT_ZERO]*measurement[i][StableData.INT_THREE])
	/measurement[i][StableData.INT_ONE];
```

```
                    literarinessDuration+=literarinessLevel[i];
                    System.out.println("literarinessLevel:" +literarinessLevel[i]);
          }
          literarinessDuration/=literarinessLevel.length;
          System.out.println("literarinessDuration:"+ literarinessDuration);
          }
public static void main(String[] argv) throws IOException {
          String text1= "  //20210727 采样数据仅保存一次，之后相同全部删除。";
//ICA kernel
double[][] kernel= new double[3][];
          kernel[0]= new InitBehaviorICAKernel().getBehaviorICAKernel(text1);
          kernel[1]= new InitBehaviorICAKernel().getBehaviorICAKernel(text2);
          kernel[2]= new InitBehaviorICAKernel().getBehaviorICAKernel(text3);
          LiterarinessLevelTest educationLevelTest= new LiterarinessLevelTest();
          educationLevelTest.getEducationLevel(kernel);
          }}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.test;
import java.io.IOException;
import org.tinos.behavior.ICA.EducationRatio;
import org.tinos.view.stable.StableData;
public class EducationLevelTest{
          double[] EducationLevel;
          double EducationDuration;
public void getEducationLevel(double[][] measurement) {
          EducationLevel= new double[measurement.length];
          for(int i= StableData.INT_ZERO;i < measurement.length;i++) {
                    EducationLevel[i]= (measurement[i][StableData.INT_THREE]+ measurement[i][StableData.INT_FOUR])
     /(measurement[i][StableData.INT_ONE]+ measurement[i][StableData.INT_TWO]
+ measurement[i][StableData.INT_THREE]+ measurement[i][StableData.INT_FOUR]
+ measurement[i][StableData.INT_FIVE]);
                    EducationDuration += EducationLevel[i];
                    System.out.println("EducationLevel:" +EducationLevel[i]);
          }
          EducationDuration/=EducationLevel.length;
          System.out.println("EducationDuration:"+ EducationDuration);
          }
public static void main(String[] argv) throws IOException {
          String text1= "  //20210727 采样数据仅保存一次，之后相同全部删除
。\r\n"
+
"  //20210727 采样数据仅保存一次，之后相同全部删除。";
//ICA kernel
double[][] kernel= new double[3][];
          kernel[0]= new EducationRatio().getEducationKernel(text1);
          kernel[1]= new EducationRatio().getEducationKernel(text2);
          kernel[2]= new EducationRatio().getEducationKernel(text3);
          //ANN kernel
          double[][] kernelRatio= new double[3][];
          kernelRatio[0]= new EducationRatio().getEducationRatioKernel(kernel[0]);
          kernelRatio[1]= new EducationRatio().getEducationRatioKernel(kernel[1]);
          kernelRatio[2]= new EducationRatio().getEducationRatioKernel(kernel[2]);
          EducationLevelTest educationLevelTest= new EducationLevelTest();
          educationLevelTest.getEducationLevel(kernelRatio);
          }}
------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.ortho.fhmm;
import java.io.IOException;
import java.util.Map;
public interface BehaviorMap{

     Map<String, String> initBusinessMap() throws IOException;
     Map<String, String> initTradeMap() throws IOException;
     Map<String, String> initFacilityMap() throws IOException;
     Map<String, String> initAckuisitionMap() throws IOException;
     Map<String, String> initCoorporationMap() throws IOException;
     Map<String, String> initPromiseMap() throws IOException;
```

```
                              }
---------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.ICA;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import org.tinos.emotion.engine.EmotionInit;
import org.tinos.emotion.engine.EnvironmentInit;
import org.tinos.emotion.estimation.EmotionSample;
import org.tinos.emotion.ortho.fhmm.EmotionMap;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.view.stable.StableData;
public class InitBehaviorICAKernel{
          private double[] kernel;
public double[] getKernel() {
                    return kernel;
                    }
public void setKernel(double[] kernel) {
                    this.kernel= kernel;
                    }
public List<String> getForRestReturn() {
                    return forRestReturn;
                    }
public void setForRestReturn(List<String> forRestReturn) {
                    this.forRestReturn= forRestReturn;
                    }
          private List<String> forRestReturn;
public double getTrustRate(String text) throws IOException {
                    EmotionInit emotionInitEnvironment= new EmotionInit();
                    emotionInitEnvironment.init(text);
                    double positiveCountEnvironment= emotionInitEnvironment.getPositiveCount();
                    double totalCountEnvironment= emotionInitEnvironment.getTotalCount();
                    positiveCountEnvironment += StableData.INT_ONE;
                    return positiveCountEnvironment/totalCountEnvironment;
                    }
public double getTrustRate(String text, Analyzer analyzer, EmotionMap emotionMap) throws IOException {
                    EmotionInit emotionInitEnvironment= new EmotionInit();
                    emotionInitEnvironment.initExcludeAnalyzer(text, analyzer, emotionMap);
                    //reduce
                    double positiveCountEnvironment= emotionInitEnvironment.getPositiveCount();
                    double totalCountEnvironment= emotionInitEnvironment.getTotalCount();
                    positiveCountEnvironment += StableData.INT_ONE;
                    return positiveCountEnvironment/totalCountEnvironment;
                    }
public double[] getBehaviorICAKernel(String text) throws IOException {
                    forRestReturn= new LinkedList<>();
                    kernel= new double[StableData.INT_SEVEN];
                    EmotionInit emotionInit= new EmotionInit();
                    emotionInit.init(text);
                    double positiveCount= emotionInit.getPositiveCount();
                    double negativeCount= emotionInit.getNegativeCount();
                    double totalCount= emotionInit.getTotalCount();
                    forRestReturn.add("正面情感:"+ positiveCount);
                    forRestReturn.add("负面情感:"+ negativeCount);
          if(positiveCount == StableData.INT_ZERO) {
                              positiveCount= StableData.INT_ONE;
                    }
          if(negativeCount == StableData.INT_ZERO) {
                              negativeCount= StableData.INT_ONE;
                    }
                    double adjRatio= Math.abs(positiveCount/negativeCount-negativeCount/positiveCount);
                    forRestReturn.add("渲染比率:"+ adjRatio);
                    double phychologicRatio= (positiveCount+ negativeCount)/totalCount;
                    forRestReturn.add("情绪比率:"+ phychologicRatio);
                    double infectionRatio= adjRatio*phychologicRatio;
```

```
                forRestReturn.add(" 感 染 比 率 :"+ infectionRatio);
                kernel[StableData.INT_ZERO]= adjRatio;
                kernel[StableData.INT_ONE]= phychologicRatio;
                kernel[StableData.INT_TWO]= infectionRatio;
                EnvironmentInit environmentInit= new EnvironmentInit();
                environmentInit.initFromEmotion(emotionInit.getWordFrequencyMap());
        Map<String, EmotionSample> environmentSampleMap= environmentInit.getEmotionSampleMap();
                forRestReturn.add("观测角度:");
                String environmentText= "";
                Iterator<String> Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getDistinction()) {
                environmentText += emotionSample.getDistinction()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(environmentText);
                kernel[StableData.INT_THREE]= getTrustRate(environmentText);
                forRestReturn.add(StableData.EMPTY_STRING+ kernel[StableData.INT_THREE]);
                forRestReturn.add("信任比率:");
                String motivationText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getMotivation()) {
                motivationText += emotionSample.getMotivation()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(motivationText);
                kernel[StableData.INT_FOUR]= getTrustRate(motivationText);
                forRestReturn.add(StableData.EMPTY_STRING+kernel[StableData.INT_FOUR]);
                forRestReturn.add("执行比率:");
                String trendingText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getTrending()) {
                trendingText += emotionSample.getTrending()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(trendingText);
                kernel[StableData.INT_FIVE]= getTrustRate(trendingText);
                forRestReturn.add(StableData.EMPTY_STRING+ kernel[StableData.INT_FIVE]);
                forRestReturn.add("成功比率:");
                String predictionText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getPrediction()) {
                predictionText += emotionSample.getPrediction()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(predictionText);
                kernel[StableData.INT_SIX]= getTrustRate(predictionText);
                forRestReturn.add(StableData.EMPTY_STRING+ kernel[StableData.INT_SIX]);
                return kernel;
                }
public double[] getBehaviorICAKernel(String text, Analyzer analyzer, EmotionMap emotionMap) throws IOException {
                forRestReturn= new LinkedList<>();
                kernel= new double[StableData.INT_SEVEN];
                EmotionInit emotionInit= new EmotionInit();
                emotionInit.initExcludeAnalyzer(text, analyzer, emotionMap);
                double positiveCount= emotionInit.getPositiveCount();
                double negativeCount= emotionInit.getNegativeCount();
                double totalCount= emotionInit.getTotalCount();
                forRestReturn.add("正面情感:"+ positiveCount);
                forRestReturn.add("负面情感:"+ negativeCount);
        if(positiveCount == StableData.INT_ZERO) {
                        positiveCount= StableData.INT_ONE;
                }
        if(negativeCount == StableData.INT_ZERO) {
                        negativeCount= StableData.INT_ONE;
                }double adjRatio= Math.abs(positiveCount/negativeCount-negativeCount/positiveCount);
                forRestReturn.add("渲染比率:"+ adjRatio);
                double phychologicRatio= (positiveCount+ negativeCount)/totalCount;
                forRestReturn.add("情绪比率:"+ phychologicRatio);
                double infectionRatio= adjRatio*phychologicRatio;
                forRestReturn.add(" 感 染 比 率 :"+ infectionRatio);
                kernel[StableData.INT_ZERO]= adjRatio;
                kernel[StableData.INT_ONE]= phychologicRatio;
                kernel[StableData.INT_TWO]= infectionRatio;
                EnvironmentInit environmentInit= new EnvironmentInit();
                environmentInit.initFromEmotionExcludeEmotion(emotionInit.getWordFrequencyMap(), emotionMap);
        Map<String, EmotionSample> environmentSampleMap= environmentInit.getEmotionSampleMap();
                forRestReturn.add("观测角度:");
                String environmentText= StableData.EMPTY_STRING;
                Iterator<String> Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getDistinction()) {
                environmentText += emotionSample.getDistinction()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(environmentText);
                kernel[StableData.INT_THREE]= getTrustRate(environmentText, analyzer, emotionMap);
                forRestReturn.add(StableData.EMPTY_STRING+kernel[StableData.INT_THREE]);
                forRestReturn.add("信任比率:");
                String motivationText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getMotivation()) {
                motivationText += emotionSample.getMotivation()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(motivationText);
                kernel[StableData.INT_FOUR]= getTrustRate(motivationText, analyzer, emotionMap);
                forRestReturn.add(StableData.EMPTY_STRING+kernel[StableData.INT_FOUR]);
                forRestReturn.add("执行比率:");
                String trendingText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null !=  emotionSample.getTrending()) {
                trendingText += emotionSample.getTrending()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(trendingText);
                kernel[StableData.INT_FIVE]= getTrustRate(trendingText, analyzer, emotionMap);
                forRestReturn.add(StableData.EMPTY_STRING+kernel[StableData.INT_FIVE]);
                forRestReturn.add("成功比率:");
                String predictionText= StableData.EMPTY_STRING;
                Iterator= environmentSampleMap.keySet().iterator();
                while(Iterator.hasNext()) {
                        String word= Iterator.next();
                        EmotionSample emotionSample= environmentSampleMap.get(word);if(null != emotionSample.getPrediction()) {
                predictionText += emotionSample.getPrediction()+ StableData.SPACE_STRING;
                        }}
                forRestReturn.add(predictionText);
                kernel[StableData.INT_SIX]= getTrustRate(predictionText, analyzer, emotionMap);
                forRestReturn.add(StableData.EMPTY_STRING+kernel[StableData.INT_SIX]);
                return kernel;
                }}
-------------------------------------------------------------------------------------------------------------------------
package org.tinos.behavior.ICA;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import org.tinos.engine.analysis.Analyzer;
import org.tinos.engine.analysis.imp.CogsBinaryForestAnalyzerImp;
import org.tinos.view.obj.WordFrequency;
import org.tinos.view.stable.StableData;
public class EducationRatio{
public double[] getEducationKernel(String text) throws IOException {
                Analyzer analyzer= new CogsBinaryForestAnalyzerImp();
                analyzer.init();
        Map<String, String> pos= analyzer.getPosCnToCn();
        List<String> sets= analyzer.parserString(text);
        Map<Integer, WordFrequency> map= analyzer.getWordFrequencyByReturnSortMap(sets);
                double[] output= new double[StableData.INT_SEVEN];
                output[StableData.INT_ZERO]= sets.size();
                Iterator<Integer> iterator= map.keySet().iterator();
                Here:
                        while(iterator.hasNext()) {
                WordFrequency wordFrequency= map.get(iterator.next());if(!pos.containsKey(wordFrequency.getWord()))            {
        continue Here;
     }if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_MING)){
        output[StableData.INT_ONE]+= StableData.INT_ONE;
        continue Here;
     }
     if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_DONG)){
        output[StableData.INT_TWO]+= StableData.INT_ONE;
        continue Here;
     }
     if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_WEI)){
        output[StableData.INT_FOUR]+= StableData.INT_ONE;
        continue Here;
     }
     if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_XING)){
        output[StableData.INT_THREE]+= StableData.INT_ONE;
        continue Here;
     }
     if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_FU)){
        output[StableData.INT_FIVE]+= StableData.INT_ONE;
        continue Here;
     }
     if(pos.get(wordFrequency.getWord()).contains(StableData.NLP_ZI_JIE)){
        output[StableData.INT_SIX]+= StableData.INT_ONE;
     }}
                return output;
        }
public double[] getEducationRatioKernel(double[] input) {
                double[] output=new double[input.length];
                for(int i=StableData.INT_ZERO;i<input.length;i++) {
                        output[i]=input[i]/input[StableData.INT_ZERO];
                }
                return output;
}}
```

具体和最新 见第十九章 元基索引函数分词 DetaParser。以及git备份溯源时间戳老版本。

Sensing等map函数的DNN RNN ANN 补充

```
package PEQ.AMV.ECS.test;
import java.io.IOException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import MSV.OSQ.sets.DetaDouble;
import OCI.ME.analysis.C.A;
import PEU.P.nlp.*;
public class DNNTest{
    public static void main(String[] argv) throws IOException, InstantiationException, IllegalAccessException {
            DNNTest dNNTest=new DNNTest();
            ANNTest aNNTest= new ANNTest();
            String[][] ann= aNNTest.getANNMatrix();
            String[][] dnn= dNNTest.getDNNMatrix(ann);
//          String[][] ann= aNNTest.getANNMatrix(string, _A);
//          String[][] dnn= dNNTest.getDNNMatrix(ann, _A, string);
            for(int i=0;i<dnn.length;i++) {
                    double dnn_lwa = DetaDouble.parseDouble(dnn[i][3]);
                    if(dnn_lwa>100) {
                            System.out.print(ann[i][0] + ":");
                            System.out.print(ann[i][1] + ":");
                            System.out.print(ann[i][2] + ":");
                            System.out.print(ann[i][3] + ":");
                            System.out.print(dnn[i][0] + ":");
                            System.out.print(dnn[i][3] + ":");
                            System.out.println("");
                    }
            }
    }

    public String[][] getDNNMatrix() throws IOException, InstantiationException, IllegalAccessException{
            ANNTest aNNTest = new ANNTest();
            RNN_IDETest rNN_IDETest = new RNN_IDETest();
            String[][] dNNMatrix = new DETA_DNN().summing_P(aNNTest.getANNMatrix(), rNN_IDETest.getIDEMatrix());
            return dNNMatrix;
    }

    public String[][] getDNNMatrix(String[][] ann) throws IOException, InstantiationException, IllegalAccessException{
            RNN_IDETest rNN_IDETest = new RNN_IDETest();
            String[][] dNNMatrix = new DETA_DNN().summing_P(ann, rNN_IDETest.getIDEMatrix());
            return dNNMatrix;
    }

    public String[][] getDNNMatrix(SensingTest sensingTest, String[][] ann, A _A, String string)
                    throws IOException, InstantiationException
    , IllegalAccessException{
            //RNN 深度此距离计算 开始注释 罗瑶光
            RNN_IDETest rNN_IDETest = new RNN_IDETest();
            String[][] rnn= rNN_IDETest.getIDEMatrixExclude_A(sensingTest, ann, _A, string);
            rnn= getPOSPCARnnMatrix(rnn);
            if(ann.length> rnn.length) {
                    ann= getAnnWithMaskRnn(ann, rnn);
            }
            String[][] dNNMatrix = new DETA_DNN().summing_P(ann, rnn);
            return dNNMatrix;
    }

    private String[][] getAnnWithMaskRnn(String[][] ann, String[][] rnn) {
            //map
            Map<String, Boolean> rnnMap= new HashMap<>();
            for(int i= 0; i< rnn.length; i++) {
                    if(!rnnMap.containsKey(rnn[i][0])) {
                            rnnMap.put(rnn[i][0], true);
                    }
            }
            String[][] maskAnn= new String[rnnMap.size()][];
            int j= 0;
            for(int i= 0;i< ann.length; i++) {
                    if(rnnMap.containsKey(ann[i][0])) {
                            maskAnn[j++]= ann[i];
                    }
            }
            return maskAnn;
    }

    private String[][] getPOSPCARnnMatrix(String[][] rnn) {
            List<String[]> list= new ArrayList<>();
            Here:
            for(int j= 0; j< rnn.length; j++) {
                            if(Double.valueOf(rnn[j][1])== 0
                              || Double.valueOf(rnn[j][1])== 0
```

```
                                || Double.valueOf(rnn[j][1])== 0
                                        ){
                                    continue Here;
                                }
                                list.add(rnn[j]);
                }
                String[][] PCANLP= new String[list.size()][15];
                Iterator<String[]> iterator= list.iterator();
                int i= 0;
                while(iterator.hasNext()) {
                        PCANLP[i++]=iterator.next();
                }
                return PCANLP;
        }
```

```
package PEQ.AMV.ECS.test;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

//import OCI.ME.analysis.C.A;
import OEI.ME.analysis.E.CogsBinaryForest_AE;
import PEU.P.nlp.*;
public class ANNTest{
   public static void main(String[] argv) throws IOException, InstantiationException, IllegalAccessException {
           ANNTest ANNTest = new ANNTest();
           String[][] AnnMatrix = ANNTest.getANNMatrix();
           for(int j = 0; j < AnnMatrix.length; j++) {
                   double sum = 0;
                   for(int i = 7; i < AnnMatrix[0].length; i++) {
                           sum += Double.valueOf(AnnMatrix[j][i]);
                   }
                   if(sum >= 0.6) {
                           System.out.println(AnnMatrix[j][0]+AnnMatrix[j][1] + AnnMatrix[j][2] + AnnMatrix[j][3] + "ANN:" + sum);
                   }
           }
   }

   public String[][] getANNMatrix() throws IOException, InstantiationException, IllegalAccessException{
           SensingTest sensingTest= new SensingTest();
           //SUM OF ANN MAP CULUMN KERNEL
           String[][] preAnnMatrix= sensingTest.getMatrix();
           String[][] AnnMatrix= new DETA_ANN_HMM().summing_P(preAnnMatrix);
           return AnnMatrix;
   }

   public String[][] getANNMatrix(String string, CogsBinaryForest_AE _A) throws IOException
   , InstantiationException, IllegalAccessException {
           SensingTest sensingTest= new SensingTest();
           //SUM OF ANN MAP CULUMN KERNEL
           String[][] preAnnMatrix= sensingTest.getMatrix(string, _A);
           String[][] AnnMatrix= new DETA_ANN_HMM().summing_P(preAnnMatrix);
           String[][] POSPCAAnnMatrix=  getPOSPCAAnnMatrix(AnnMatrix, _A.getPosCnToCn());
           return POSPCAAnnMatrix;
//         return POSPCAAnnMatrix= AnnMatrix;
   }
   public String[][] getPOSPCAAnnMatrix(String[][] AnnMatrix, Map<String, String> pos){
           List<String[]> list= new ArrayList<>();
           for(int j= 0; j< AnnMatrix.length; j++) {
                   if(pos.containsKey(AnnMatrix[j][0])) {
                           String string= pos.get(AnnMatrix[j][0]);
                           if(string.contains("名")
                                           || string.contains("动")
                                           || string.contains("医")
                                           || string.contains("谓")
                                           || string.contains("形")){
                                   list.add(AnnMatrix[j]);
```

```
                        }
                }
        }
        String[][] PCANLP= new String[list.size()][15];
        Iterator<String[]> iterator= list.iterator();
        int i= 0;
        while(iterator.hasNext()) {
                PCANLP[i++]=iterator.next();
        }
        return PCANLP;
    }

    public String[][] getANNMatrix(SensingTest sensingTest, String string, CogsBinaryForest_AE _A)
                throws IOException, InstantiationException, IllegalAccessException {
        //SUM OF ANN MAP CULUMN KERNEL
        String[][] preAnnMatrix = sensingTest.getMatrix(string, _A);
        String[][] AnnMatrix = new DETA_ANN_HMM().summing_P(preAnnMatrix);
        String[][] POSPCAAnnMatrix=  getPOSPCAAnnMatrix(AnnMatrix, _A.getPosCnToCn());
        return POSPCAAnnMatrix;
    }
}
```

```
package PEQ.AMV.ECS.test;
import java.io.IOException;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import OCI.ME.analysis.C.A;
import OEI.ME.analysis.E.CogsBinaryForest_AE;
public class RNN_IDETest{
    public static void main(String[] argv) throws IOException, InstantiationException, IllegalAccessException {
        RNN_IDETest rNN_IDETest= new RNN_IDETest();
        rNN_IDETest.getIDEMatrix();
    }

    public String[][] getIDEMatrix() throws IOException, InstantiationException, IllegalAccessException{
        SensingTest sensingTest = new SensingTest();
        String[][] sensingMatrix = sensingTest.getMatrix();
        Map<String, List<Double>> map = new HashMap<>();
        for(int i = 0; i < sensingMatrix.length; i++) {
                List<Double> list = new LinkedList<>();
                list.add(1.0);
                map.put(sensingMatrix[i][0], list);
        }
        String[][] ideMatrix = new String [sensingMatrix.length][4];
        List<String> sets = sensingTest.getSets();
        Map<String, String> pos = sensingTest.getPosCnToCn();
        Iterator<String> setsIterator = sets.iterator();
        double count = 1;
        //map position
        while(setsIterator.hasNext()) {
                String word = setsIterator.next();
                if(map.containsKey(word)) {
                        List<Double> list  = map.get(word);
                        list.add(count);
                        map.put(word, list);
                }
                count++;
        }
        //RNN LOOP position
        int ideMatrixCount = 0;
        Iterator<String> mapIterator = map.keySet().iterator();
        while(mapIterator.hasNext()) {
                String word = mapIterator.next();
                List<Double> list = map.get(word);
                double dovFactor = 1;
                double popFactor = 0;
                double eopFactor = 1;
                double dovCount = 1;
```

```
			for(int i = 0; i < list.size(); i++) {
				for(int j = i + 1; j < list.size(); j++) {
					dovCount ++;
					dovFactor += list.get(j);
				}
				dovFactor += Math.abs(list.get(i) - dovFactor);
				eopFactor += (eopFactor + list.get(i)) / 2;
			}
			//pos normalization
			if(pos.containsKey(word)) {
				popFactor += pos.get(word).contains("动")? 16: 0;
				popFactor += pos.get(word).contains("名")? 4: 0;
				popFactor += pos.get(word).contains("形")? 2: 0;
			}
			ideMatrix[ideMatrixCount][0] = word;
			ideMatrix[ideMatrixCount][1] = "" + popFactor;
			ideMatrix[ideMatrixCount][2] = "" + dovFactor/dovCount;
			ideMatrix[ideMatrixCount][3] = "" + eopFactor;
			ideMatrixCount++;
		}
		return ideMatrix;
	}

	public String[][] getIDEMatrixExclude_A(CogsBinaryForest_AE _A, String string) throws IOException {
		SensingTest sensingTest = new SensingTest();
		String[][] sensingMatrix = sensingTest.getMatrix(string, _A);
		Map<String, List<Double>> map = new HashMap<>();
		for(int i = 0; i < sensingMatrix.length; i++) {
			List<Double> list = new LinkedList<>();
			list.add(1.0);
			map.put(sensingMatrix[i][0], list);
		}
		String[][] ideMatrix = new String [sensingMatrix.length][4];
		List<String> sets = sensingTest.getSets();
		Map<String, String> pos = sensingTest.getPosCnToCn();
		Iterator<String> setsIterator = sets.iterator();
		double count = 1;
		//map position
		while(setsIterator.hasNext()) {
			String word = setsIterator.next();
			if(map.containsKey(word)) {
				List<Double> list  = map.get(word);
				list.add(count);
				map.put(word, list);
			}
			count++;
		}
		//RNN LOOP position
		int ideMatrixCount = 0;
		Iterator<String> mapIterator = map.keySet().iterator();
		while(mapIterator.hasNext()) {
			String word = mapIterator.next();
			List<Double> list = map.get(word);
			double dovFactor = 1;
			double popFactor = 0;
			double eopFactor = 1;
			double dovCount = 1;
			for(int i = 0; i < list.size(); i++) {
				for(int j = i + 1; j < list.size(); j++) {
					dovCount ++;
					dovFactor += list.get(j);
				}
				dovFactor += Math.abs(list.get(i) - dovFactor);
				eopFactor += (eopFactor + list.get(i)) / 2;
			}
			//pos normalization
			if(pos.containsKey(word)) {
				popFactor += pos.get(word).contains("动")? 16: 0;
				popFactor += pos.get(word).contains("名")? 4: 0;
				popFactor += pos.get(word).contains("形")? 2: 0;
			}

			ideMatrix[ideMatrixCount][0] = word;
			ideMatrix[ideMatrixCount][1] = "" + popFactor;
```

```
				ideMatrix[ideMatrixCount][2] = "" + dovFactor/dovCount;
				ideMatrix[ideMatrixCount][3] = "" + eopFactor;
				ideMatrixCount++;
			}
			return ideMatrix;
	}

	public String[][] getIDEMatrixExclude_A(SensingTest sensingTest, String[][] ann, A _A, String string) {
			//敏感度 意识 sensing
			String[][] sensingMatrix = ann;
			Map<String, List<Double>> map = new HashMap<>();
			for(int i = 0; i < sensingMatrix.length; i++) {
				List<Double> list = new LinkedList<>();
				list.add(1.0);
				map.put(sensingMatrix[i][0], list);
			}
			String[][] ideMatrix= new String [sensingMatrix.length][4];
			List<String> sets= sensingTest.getSets();
			Map<String, String> pos= sensingTest.getPosCnToCn();
			Iterator<String> setsIterator= sets.iterator();
			double count= 1;
			//map position
			//开始计算图距离
			while(setsIterator.hasNext()) {
				String word= setsIterator.next();
				if(map.containsKey(word)) {
					List<Double> list= map.get(word);
					list.add(count);
					map.put(word, list);
				}
				count++;
			}
			//RNN LOOP position
			int ideMatrixCount= 0;
			Iterator<String> mapIterator= map.keySet().iterator();
			while(mapIterator.hasNext()) {
				String word= mapIterator.next();
				List<Double> list= map.get(word);
				double dovFactor= 1; //距离distance的距离 distance of same vebals
				double popFactor= 0; // 语义距离 part of speech
				double eopFactor= 1; // 位移距离 eclid of parts 我全部会注释 到处是猫腻
				double dovCount= 1; // 计数
				for(int i= 0; i< list.size(); i++) {
					for(int j= i + 1; j< list.size(); j++) {
						dovCount++;
						dovFactor+= list.get(j);
					}
					dovFactor+= Math.abs(list.get(i)- dovFactor); //平方和距离开方
					eopFactor+= (eopFactor+ list.get(i))/ 2;// 欧基里德用来计算熵增
				}
				//pos normalization
				if(pos.containsKey(word)) {
					popFactor+= pos.get(word).contains("名")? 16: 0;
					popFactor+= pos.get(word).contains("动")? 5: 0;
					popFactor+= pos.get(word).contains("医")? 3: 0;
					popFactor+= pos.get(word).contains("谓")? 3: 0;
					popFactor+= pos.get(word).contains("形")? 2: 0;
				}
				ideMatrix[ideMatrixCount][0]= word;
				ideMatrix[ideMatrixCount][1]= ""+ popFactor;
				ideMatrix[ideMatrixCount][2]= ""+ dovFactor/ dovCount;// 平均distance数
				ideMatrix[ideMatrixCount][3]= ""+ eopFactor;
				ideMatrixCount++;
			}
			//这里再2018年 最老的版本我用的是system来进行print, 后来包装成函数我就注释掉了, 后来优化就删去了system 的打印函数,
罗瑶光20210420
			return ideMatrix;
	}
}
```