目录

[第一章 德塔自然语言图灵系统 17](#_Toc32678)

[第一节 研发说明 17](#_Toc20456)

[德塔语言图灵工程API说明书V\_10\_6\_1 17](#_Toc18195)

[起源动机 17](#_Toc17068)

[简介 17](#_Toc8639)

[使用方法 17](#_Toc28510)

[具体重要功能展示 21](#_Toc25770)

[适用范围 22](#_Toc20028)

[注意 22](#_Toc18769)

[感谢 23](#_Toc13349)

[研发需要清单 23](#_Toc5412)

[第二节 研发笔记 23](#_Toc11428)

[德塔读心术词汇重心图算法思想手稿20190908 23](#_Toc6615)

[第三节 研发源码 27](#_Toc19863)

[StableData, 静态类 27](#_Toc15615)

[FMHMMNode, 隐马尔可夫类 50](#_Toc6331)

[WordFrequency, 词频类 51](#_Toc27101)

[FMHMMPOS, 隐马尔可夫类 51](#_Toc25313)

[FMHMMNode, 隐马尔可夫类 52](#_Toc9802)

[Verbal, 词汇处理类 52](#_Toc13729)

[DemoPOSforSpecial, Demo类 53](#_Toc3093)

[DemoPOS, Demo类 54](#_Toc1983)

[FHMMList, 隐马尔可夫类 56](#_Toc12893)

[PillowsSet, 资源类 57](#_Toc18850)

[FMHMMListOneTimeImp, 隐马尔可夫类 61](#_Toc11944)

[FMHMMListImp, 隐马尔可夫类 102](#_Toc28295)

[POSControllerImp, 语义处理类 107](#_Toc21447)

[POSControllerCognitionImp, 语义处理类 129](#_Toc8285)

[NLPController, 自然语言处理类 132](#_Toc17610)

[NLPControllerImp, 自然语言处理类 133](#_Toc31593)

[NEROController, 神经网络索引类 143](#_Toc25736)

[NEROControllerOneTimeImp, 神经网络索引类 143](#_Toc22565)

[NEROControllerImp, 神经网络索引类 145](#_Toc11921)

[Quick6DLuoYaoguangSortImp, 极快速排序类 146](#_Toc7413)

[Quick6DLuoYaoguangSort3DMapImp, 极快速排序类 148](#_Toc11393)

[EuclidControllerImp, 欧基里德算法类 149](#_Toc7847)

[EuclidController, 欧基里德算法类 150](#_Toc4871)

[Analyzer, 分词类 151](#_Toc3549)

[AnalyzerImp, 分词类 151](#_Toc11304)

[WordFrequencyUtil, 词频类 157](#_Toc8397)

[WordForestUtil, 索引森林类 158](#_Toc8318)

[SensingMapImp, 意识类 159](#_Toc25597)

[TranslatorImp, 翻译类 160](#_Toc32640)

[LenovoTest, 联想类 168](#_Toc12771)

[EnvironmentTest, 环境类 169](#_Toc10019)

[EmotionTest, 思维类 170](#_Toc4248)

[EnvironmentInit, 环境类 171](#_Toc27977)

[EmotionInit, 思维类 175](#_Toc28153)

[LenovoInit, 联想类 178](#_Toc16854)

[RatioMapImp, 比率图类 179](#_Toc13797)

[SuccessICATest, 比率分析类 184](#_Toc30323)

[LiterarinessLevelTest, 文学类 185](#_Toc23432)

[EducationLevelTest, 教育类 185](#_Toc313)

[InitBehaviorICAKernel, 习惯类 186](#_Toc30154)

[EducationRatio, 教育类 191](#_Toc31193)

[第四节 DNA元基索引版本等 192](#_Toc20568)

[Sensing等maptest的过程函数补充 192](#_Toc19719)

[DETARNN\_IDETEST, 卷积类 196](#_Toc7655)

[DETAANNTEST, 卷积类 200](#_Toc16428)

[DETADNNTEST, 卷积类 201](#_Toc7584)

[第二章 Java数据分析算法引擎系统 204](#_Toc28584)

[第一节 研发说明 204](#_Toc21718)

[德塔Java数据分析算法引擎系统说明书V\_1\_0\_2 204](#_Toc1069)

[起源动机 204](#_Toc18288)

[应用特色 204](#_Toc2)

[使用方法 204](#_Toc11441)

[功能注解 204](#_Toc19232)

[适用范围 204](#_Toc19276)

[注意 204](#_Toc7569)

[感谢 204](#_Toc16085)

[研发需要清单 204](#_Toc8573)

[第二节 研发笔记 204](#_Toc14873)

[API运行原理Flowchat 204](#_Toc12986)

[罗瑶光小高峰过滤快排4代原理 204](#_Toc13542)

[两种比较领先的排序思维对比 204](#_Toc4072)

[维度卷积计算原理 204](#_Toc10558)

[第三节 研发源码 204](#_Toc12636)

[CheckRange, 区间类 204](#_Toc25364)

[Closing, 卷积类 204](#_Toc23651)

[Dilation, 卷积类 204](#_Toc16131)

[Emboss, 卷积类 204](#_Toc32169)

[Erosion, 卷积类 204](#_Toc22428)

[GetMean, 卷积类 204](#_Toc27587)

[Guassian, 卷积类 204](#_Toc24124)

[HitAndMiss, 卷积类 204](#_Toc31540)

[HoughTransform, 卷积类 204](#_Toc913)

[Laplacian, 卷积类 204](#_Toc22368)

[Mask, 卷积类 204](#_Toc13612)

[Median, 卷积类 204](#_Toc28746)

[Opening, 卷积类 204](#_Toc24796)

[ReadWritePng, 卷积类 204](#_Toc2833)

[RegionGet, 卷积类 204](#_Toc13220)

[Sobel, 卷积类 204](#_Toc13190)

[Strech, 卷积类 204](#_Toc14722)

[Threshold, 卷积类 204](#_Toc32600)

[CnnMeasure, 卷积类 204](#_Toc11371)

[DETA\_ANN\_HMM, 卷积类 204](#_Toc22432)

[DETA\_DNN, 卷积类 204](#_Toc29733)

[BinarySearch, 排序搜索类 204](#_Toc25561)

[BreadthRun, 排序搜索类 204](#_Toc9698)

[BreadthTreeSearch, 排序搜索类 204](#_Toc18822)

[DepthRun, 排序搜索类 204](#_Toc2510)

[DepthTreeSearch, 排序搜索类 204](#_Toc19985)

[LinerSearch, 排序搜索类 204](#_Toc7656)

[PreorderRun, 排序搜索类 204](#_Toc5294)

[PreorderTreeSearch, 排序搜索类 204](#_Toc2281)

[RandomSearch, 排序搜索类 204](#_Toc16452)

[BinarySort, 排序搜索类 205](#_Toc17660)

[BTreeSort, 排序搜索类 205](#_Toc16810)

[Heap\_1D\_Sort, 排序搜索类 205](#_Toc18833)

[InsertionSort, 排序搜索类 205](#_Toc7351)

[Leaf, 排序搜索类 205](#_Toc29857)

[LinkSort, 排序搜索类 205](#_Toc765)

[LYGSort, 排序搜索类 205](#_Toc14094)

[OrderEvenSort, 排序搜索类 205](#_Toc22816)

[OTreeSort, 排序搜索类 205](#_Toc25863)

[Quick\_5D\_Sort, 排序搜索类 205](#_Toc9261)

[Quick\_6D\_luoyaoguang\_Sort, 排序搜索类 205](#_Toc21148)

[Quick\_Luoyaoguang\_4D, 排序搜索类 205](#_Toc1627)

[SelectionSort, 排序搜索类 205](#_Toc26087)

[TTreeSort, 排序搜索类 205](#_Toc28441)

[LineCodeOperation, 计算类 205](#_Toc21358)

[PixFloat, 像素处理类 205](#_Toc94)

[Copy, 复制类 205](#_Toc7855)

[DFT, 卷积类 205](#_Toc31629)

[Laplasian, 卷积类 205](#_Toc2035)

[MaxAndMin, 卷积类 205](#_Toc23151)

[Median, 卷积类 205](#_Toc6623)

[PeakStatistic, 统计类 205](#_Toc7444)

[Proportion, 统计类 205](#_Toc23640)

[Quantification, 统计类 205](#_Toc23445)

[Shehold, 卷积类 205](#_Toc8115)

[Tailor, 统计类 205](#_Toc25357)

[第四节 DNA元基索引版本略，养疗经音频时序波的处理应用示例 205](#_Toc10830)

[第三章 德塔ETL人工智能可视化数据流分析引擎系统 205](#_Toc8012)

[第一节 研发说明 205](#_Toc9218)

[德塔ETL可视化数据分析引擎系统说明书说明书 205](#_Toc24321)

[起源动机 205](#_Toc13837)

[简介 205](#_Toc11900)

[使用方法 205](#_Toc19302)

[具体重要功能展示 205](#_Toc9524)

[档案管理功能 205](#_Toc31897)

[流操作中相同逻辑节点重用功能 205](#_Toc9388)

[流操作节点配置功能 205](#_Toc19357)

[异常消息面板 205](#_Toc15226)

[适用范围 205](#_Toc14148)

[注意 205](#_Toc13683)

[感谢 205](#_Toc11055)

[研发需要清单 205](#_Toc27212)

[第二节 研发源码 205](#_Toc14452)

[SaveAndUpdateFile, 文件类 205](#_Toc9776)

[SaveAsANewFile, 文件类 205](#_Toc9297)

[OSGI\_chansfer, 接口类 205](#_Toc9243)

[OSGI\_rigester, 接口类 205](#_Toc1446)

[GUIsample, ETL引擎类 205](#_Toc4091)

[ThisCanvas, 画图引擎类 206](#_Toc12042)

[CheckRange, 区间类 206](#_Toc30436)

[DrawArrow, 画图类 206](#_Toc21171)

[DrawFlashSide, 画图类 206](#_Toc32220)

[DrawSinLine, 画图类 206](#_Toc30356)

[DynamicLineUpdater, 动态画图类 206](#_Toc15365)

[LinkList, 链表类 206](#_Toc16978)

[LinkNode, 链表类 206](#_Toc15162)

[UpdateRelatedLine, 画图类 206](#_Toc15877)

[NodeInfo, 链表类 206](#_Toc31311)

[NodeProject, 链表类 206](#_Toc4063)

[MyPanel, 画图类 206](#_Toc26262)

[CacuString, 字符处理类 206](#_Toc30767)

[NodeShow, 节点类 206](#_Toc11868)

[LinkOSGI, 接口类 206](#_Toc7289)

[NodeOSGI, 接口类 206](#_Toc5828)

[ObjectInterface, 插件接口类 206](#_Toc27939)

[ObjectPanel, 插件接口类 206](#_Toc26803)

[ObjectRun, 插件接口类 206](#_Toc20138)

[ObjectView, 插件接口类 206](#_Toc21694)

[DrawArrowHead, 画图类 206](#_Toc5321)

[DrawNeroCellMask31, 画图类 206](#_Toc12101)

[DrawNeroCellMask32, 画图类 206](#_Toc7077)

[UnicornJSplitPane, 组件类 206](#_Toc32765)

[UnicornSplitPaneUI, 组件类 206](#_Toc32699)

[UnicornTreeCellRenderer, 组件类 206](#_Toc16032)

[UnicornTreeUI, 组件类 206](#_Toc14921)

[DeleteFile, 文件类 206](#_Toc1586)

[arffTransferNodeInterface, 插件类 206](#_Toc31627)

[arffTransferNodePanel, 插件类 206](#_Toc19921)

[arffTransferNodeRun, 插件类 206](#_Toc18685)

[arffTransferNodeView, 插件类 206](#_Toc6325)

[arffNode, 节点类 206](#_Toc24650)

[arffLink, 节点类 206](#_Toc13829)

[DrawNeroCellMask33, 画图类 206](#_Toc32082)

[第三节 DNA元基索引版本 206](#_Toc9244)

[ThisCanvas去画面闪烁机制 206](#_Toc30492)

[第四章 德塔Socket流可编程数据库语言引擎系统 206](#_Toc12209)

[第一节 研发说明 206](#_Toc6689)

[德塔Socket流可编程数据库语言引擎系统 206](#_Toc25460)

[起源动机 206](#_Toc24072)

[简介 206](#_Toc22871)

[使用方法 206](#_Toc6710)

[具体重要功能展示 206](#_Toc9063)

[适用范围 206](#_Toc14297)

[注意 206](#_Toc28705)

[感谢 206](#_Toc5470)

[研发需要清单 206](#_Toc28318)

[第二节 研发笔记 207](#_Toc11880)

[DETASocketPLSQLDatabaseFramework7 207](#_Toc22809)

[DETADatabasePLSQL 207](#_Toc29899)

[DETAPLSQLCommands 207](#_Toc146)

[CommandsDefinition 207](#_Toc27161)

[CommandSamples 207](#_Toc19100)

[RealWorldSamplesByUsingDETAPLSQLDatabase 207](#_Toc18105)

[Acknowledgement 207](#_Toc23199)

[Logbin 207](#_Toc8872)

[第三节 研发源码 207](#_Toc10764)

[ConfigController, 控制类 207](#_Toc30034)

[DBCategoryController, 控制类 207](#_Toc12123)

[DeleteController, 控制类 207](#_Toc11558)

[InsertController, 控制类 207](#_Toc29126)

[SelectController, 控制类 207](#_Toc25807)

[UpdateController, 控制类 207](#_Toc6181)

[VPC, 事务类 207](#_Toc11070)

[BootVPCS, 事务类 207](#_Toc13140)

[RequestFilterController, 控制类 207](#_Toc15585)

[RequestFixController, 控制类 207](#_Toc18689)

[RequestRecordController, 控制类 207](#_Toc16317)

[ResponseController, 控制类 207](#_Toc2531)

[ServerInitController, 控制类 207](#_Toc24911)

[Sleeper, 事物类 207](#_Toc13987)

[SleeperHall, 事务类 207](#_Toc27730)

[ForwardVision, 事务类 207](#_Toc4343)

[RestMapVision, 事务类 207](#_Toc24032)

[VPCSResponse, 事务类 207](#_Toc25289)

[DatabaseLogHall, 日志类 207](#_Toc31889)

[DetaCacheManager, 日志类 207](#_Toc13149)

[DetaDBBufferCacheManager, 数据类 207](#_Toc18811)

[DetaDBUtil, 数据类 207](#_Toc17102)

[DetaUtil, 数据类 207](#_Toc8766)

[CreateTablesImp, 数据类 207](#_Toc26210)

[DeleteRowsImp, 数据类 207](#_Toc25741)

[InsertRowsImp, 数据类 207](#_Toc29820)

[ExecPLSQLImp, 数据类 207](#_Toc9929)

[PLSQLCommandImp, 数据类 207](#_Toc9538)

[ProcessAggregationPLSQL, 数据类 207](#_Toc30379)

[ProcessConditionPLSQL, 数据类 207](#_Toc13603)

[ProcessGetCulumnsPLSQL, 数据类 207](#_Toc22037)

[ProcessRelationPLSQL, 数据类 207](#_Toc16611)

[Cell, 数据类 207](#_Toc12153)

[SelectJoinRowsImp, 数据类 207](#_Toc20199)

[SelectNestRowsImp, 数据类 207](#_Toc1661)

[SelectRowsImp, 数据类 207](#_Toc22467)

[UpdateJoinRowsImp, 数据类 207](#_Toc3786)

[UpdateRowsImp, 数据类 207](#_Toc28766)

[LoginServiceImpl, 数据类 208](#_Toc25113)

[LoginDAOImpl, 数据类 208](#_Toc2607)

[RestControllerPortImpl, 控制类 208](#_Toc6152)

[RestDBConfigImpl，WEB接口类 208](#_Toc8795)

[RestDBDeleteImpl, WEB接口类 208](#_Toc21821)

[RestDBInsertImpl, WEB接口类 208](#_Toc16055)

[RestDBPLSQLImpl, WEB接口类 208](#_Toc21719)

[RestDBSelectImpl, WEB接口类 208](#_Toc20484)

[RestDBUpdateImpl, WEB接口类 208](#_Toc17987)

[RestLoginPortImpl, WEB接口类 208](#_Toc2782)

[TransactionDelegate, 事务类 208](#_Toc27381)

[第四节 VPCSStandard函数提取，养疗经服务器群的继承函数太多，千篇一律略。 208](#_Toc5419)

[ServerForward\_Standard, 标准数据类 208](#_Toc31371)

[ServerInit\_Standard, 标准数据类 208](#_Toc5142)

[ServerRestMap\_Standard, 标准数据类 208](#_Toc18788)

[ServerSleeper\_Standard, 标准数据类 208](#_Toc18139)

[ServerVPC\_Standard, 标准数据类 208](#_Toc19647)

[VPCSRequest, 标准数据类 208](#_Toc31979)

[VPCSResponse, 标准数据类 208](#_Toc12492)

[第五章 德塔数据结构变量快速转换 208](#_Toc27184)

[第一节 研发说明 208](#_Toc6236)

[起源动机 208](#_Toc12926)

[简介 208](#_Toc6939)

[使用方法 208](#_Toc3436)

[适用范围 208](#_Toc32366)

[注意 208](#_Toc11829)

[感谢 208](#_Toc18030)

[研发需要清单 208](#_Toc14152)

[第二节 研发笔记 208](#_Toc26523)

[DetaDataSwapDetailsMapV1.0 208](#_Toc32689)

[德塔数据结构变量快速转换引擎系统复杂点解析1.0 208](#_Toc26420)

[API使用流程FlowChat 208](#_Toc24712)

[混合数列排序FlowChat 208](#_Toc26466)

[第三节 研发源码 208](#_Toc13056)

[ArraySwap, 数组类 208](#_Toc17808)

[ArrayValidation, 数组类 208](#_Toc11218)

[CSVSwap, Office类9 208](#_Toc25765)

[DateSwap, 时间类 208](#_Toc26388)

[DateValidation, 时间类 208](#_Toc18944)

[HashSwap, 哈希类 208](#_Toc20416)

[HttpUnicode, WEB类 208](#_Toc11109)

[ImageSwap, 图片类 208](#_Toc10356)

[IteratorSwap, heap类 208](#_Toc15677)

[JsonSwap, 字码类 208](#_Toc19406)

[ListSwap, 链表类 208](#_Toc25650)

[ListValidation, 链表类 208](#_Toc16226)

[MapSwap, 图类 208](#_Toc7148)

[MatrixSwap, 矩阵类 208](#_Toc27251)

[MatrixValidation, 矩阵类 209](#_Toc1821)

[Matrix3DSwap, 矩阵类 209](#_Toc14927)

[ObjectSwap, 对象类 209](#_Toc24406)

[StockCode, 股市类 209](#_Toc26074)

[QuickLuoyaoguang4D, 排序类 209](#_Toc7479)

[StringSequency, 字符频率类 209](#_Toc21328)

[StringValidation, 字符类 209](#_Toc9637)

[StringSwap, 字符类 209](#_Toc30011)

[StringBuilderSwap, 字符类 209](#_Toc1386)

[TreeSwap, 图类 209](#_Toc8186)

[TSP, 商旅类 209](#_Toc22308)

[TSPEuler, 商旅类 209](#_Toc9349)

[YaoguangEulerTSP, 商旅类 209](#_Toc3446)

[TXTSwap, 文本类 209](#_Toc8790)

[VectorSwap, 向量类 209](#_Toc27594)

[XMLSwap, 脚本类 209](#_Toc20898)

[第四节 DNA元基索引版本 209](#_Toc2606)

[第六章 数据预测引擎系统 209](#_Toc20546)

[第一节 研发说明 209](#_Toc25871)

[起源动机 209](#_Toc3836)

[数据预测引擎系统说明书说明书 209](#_Toc16356)

[简介 209](#_Toc10771)

[使用方法 209](#_Toc6655)

[具体重要功能展示 209](#_Toc3904)

[档案管理功能 209](#_Toc12066)

[动态识别眼睛例子 209](#_Toc9343)

[算法搜索的NLP匹配打分 209](#_Toc2445)

[适用范围 209](#_Toc32354)

[注意 209](#_Toc6555)

[感谢 209](#_Toc27067)

[研发需要清单 209](#_Toc10025)

[第二节 研发笔记 209](#_Toc27617)

[关于核心算法欧拉森林商旅路径思想解析 209](#_Toc21344)

[关于核心算法坐标团重心轨迹算法思想图解 209](#_Toc13792)

[关于核心算法压强斥力和运动轨迹思想 209](#_Toc21909)

[关于核心算法切裂算法思想导图 209](#_Toc10828)

[关于核心算法内部分子相互斥力计算思想 209](#_Toc13399)

[关于核心算法雷达边缘路径计算思想 209](#_Toc21139)

[第三节 研发源码 209](#_Toc1012)

[Fissile, 坐标裂类 209](#_Toc18437)

[FissileWithMatch, 坐标裂类 209](#_Toc15850)

[FuzzProbabailityClassification, 分类类 209](#_Toc30181)

[PositionClasification, 分类类 209](#_Toc8484)

[PositionHeartsSample, 坐标类 209](#_Toc1349)

[ProbabilityClasification, 概率分类类 209](#_Toc18684)

[ClusterAttraction, 簇类 209](#_Toc2168)

[Fusion, 融类 209](#_Toc22861)

[FusionHeart, 融坐标类第517页 209](#_Toc27488)

[SideEnd, 边坐标类 210](#_Toc20513)

[FindHeartPositions, 坐标类 210](#_Toc5860)

[FindMidPositions, 坐标类 210](#_Toc20922)

[ErrorAsserts, 检测类 210](#_Toc26954)

[PositionsHintDirection, 隐坐标类 210](#_Toc10766)

[CorrelationICA, 分析类 210](#_Toc29272)

[ImagePixClassification, 像素类 210](#_Toc22248)

[ImagePixExtract, 像素类 210](#_Toc6490)

[ImagePixGroupFilter, 像素类 210](#_Toc18510)

[ForestIsolation, 坐标观察类 210](#_Toc17541)

[Isolation, 观察类 210](#_Toc12509)

[MatrixIsolationFilter, 观察类 210](#_Toc5199)

[IsIssueDate, 检测类 210](#_Toc20023)

[NLPTopicMatch, 语言处理类 210](#_Toc6839)

[FindPCAMeanDistance, 分析类 210](#_Toc4539)

[FindPositionsGroupPascalHearts, 坐标趋势类 210](#_Toc30097)

[FusionPCAFilter, 过滤类 210](#_Toc5149)

[PCAMeanOfFuzzPC, 分析类 210](#_Toc11333)

[PCAPositionFilter, 过滤类 210](#_Toc21141)

[DistanceRatio, 比率类 210](#_Toc29881)

[LYG4DWithDoubleQuickSort4D, 排序类 210](#_Toc27333)

[TraceFissilePositionHearts, 轨迹类 210](#_Toc24620)

[TracePositionHearts, 轨迹类 210](#_Toc29313)

[YaoguangLuoEulerRingTSP, 商旅类 210](#_Toc14418)

[YaoguangLuoEulerRingTSP2D, 商旅类 210](#_Toc13084)

[第四节 DNA元基索引版本 210](#_Toc22007)

[TraceFissile\_AMV\_MVS\_VSQ\_ByHearts, 商旅类 210](#_Toc3055)

[第七章 类人DNA与神经元基于催化算子映射编码方式 210](#_Toc17889)

[1 DETA humanoid cognition 210](#_Toc7721)

[1.1DETAhumanoidcognitionhistory, 德塔类人认知历史 210](#_Toc30178)

[1.2DETAhumanoidcognitiondevelopment, 德塔类人认知研发 210](#_Toc13369)

[1.3 DETA humanoid cognition application, 德塔类人认知应用 210](#_Toc13092)

[2 DETA Business back end logic 210](#_Toc10561)

[2.1DETABusinessbackendlogichistory, 德塔商业后端逻辑历史 210](#_Toc19518)

[2.2DETABusinessbackendlogicdevelopment, 德塔商业后端逻辑发展 210](#_Toc31099)

[2.3DETABusinessbackendlogicapplication, 德塔商业后端逻辑应用 210](#_Toc31896)

[3 DETA Catalytic computing 210](#_Toc10874)

[3.1DETACatalyticcomputinghistory, 德塔催化计算历史 210](#_Toc13335)

[3.2DETACatalyticcomputingdevelopment, 德塔催化计算发展 210](#_Toc29348)

[3.3DETACatalyticcomputingapplication, 德塔催化计算应用 210](#_Toc9533)

[4 DETA Finding initions 210](#_Toc354)

[4.1DETAFindinginitionshistory, 德塔催化计算算子单元寻找历史 210](#_Toc5397)

[4.2DETAFindinginitionsdevelopment, 德塔催化计算算子单元寻找发展 210](#_Toc5581)

[4.3DETAFindinginitionsapplication, 德塔催化计算算子单元寻找应用 210](#_Toc925)

[5 DETA DNA decoding 210](#_Toc16368)

[5.1DETADNAdecodinghistory, 德塔催化单元的DNA解码历史 210](#_Toc14496)

[5.2DETADNAdecodingdevelopment, 德塔催化单元的DNA解码发展 210](#_Toc18634)

[5.3DETADNAdecodingapplication, 德塔催化单元的DNA解码应用 210](#_Toc24587)

[6 IDUC DNA and Its Applications, IDUC DNA与它的应用 211](#_Toc24506)

[7 IDUC VPCS AOPM 3D Nero Cell and Its Applications, 3维神经建模与应用 211](#_Toc31303)

[8 Refer 211](#_Toc24575)

[第八章 肽展公式推导与元基编码进化计算以及它的应用发现 211](#_Toc18473)

[1 DETA INITONS classify/德塔元基分类 211](#_Toc22377)

[2 DETA INITONS PDN words root/德塔元基分类词根 211](#_Toc22371)

[3 DETA INITONS PDN words/德塔元基分类词典 211](#_Toc32629)

[4 DETA TVM/德塔词典肽翻译虚拟机 211](#_Toc15277)

[5 DETA TVM applications/德塔肽翻译虚拟机应用技术 211](#_Toc9322)

[6 DETA TVM PDC/虚拟机应用优化 211](#_Toc8695)

[7 DETA TVM PDE/德塔肽翻译推导 211](#_Toc4654)

[8 DETA TVM PDC functions/德塔肽推导函数化 211](#_Toc5732)

[9 DETA TVM PDC function optimization and PDE/德塔肽推导函数逻辑优化 211](#_Toc7399)

[10 DETA TVM PDE Logic/德塔肽推导函数逻辑优化成肽展公式化 211](#_Toc23259)

[11 DETA TVM PDE and its application/德塔肽展公式应用论证技术 211](#_Toc19381)

[12 TVM humanoid life Research/应用在类人生命进化中 211](#_Toc23415)

[13 Eternal Research/应用在类人生命永生探索领域 211](#_Toc7670)

[14 Not the End/似乎刚刚开始… 211](#_Toc30490)

[15 Conclusion 211](#_Toc26616)

[16 Reference 211](#_Toc17901)

[17 Thanks 211](#_Toc27152)

[第九章 DNA催化与肽展计算和AOPM-TXH-VECS-IDUQ元基解码 211](#_Toc5272)

[1.推导与定义:甲基胞嘧啶在DNA编码和肽计算中具体定义为IDUQ-U变嘧啶 211](#_Toc7514)

[2.推导与定义:2氨基腺嘌呤在DNA编码和肽计算中具体定义为VECS-V变感腺嘌呤 211](#_Toc17009)

[3.推导与定义:次黄嘌呤在DNA编码和肽计算中具体定义为VECS-E尿变嘌呤 211](#_Toc19396)

[4.推导与定义:AOPM-A变胸腺苷, AOPM-O尿胞变腺苷, AOPM-P尿胞变鸟苷](#_Toc25080)

[, AOPM-M鸟腺苷的S形螺旋纹血氧峰触发器分子式催化计算严谨完整过程 211](#_Toc25080)

[5.推导与定义:VECS-VECS嘌呤对, VECS嘌呤弧, VECS-IDUQ碱基对, IDUQ-IDUQ嘧啶对的催化模型 211](#_Toc29957)

[6.推导与定义:次黄嘌呤, 尿变嘌呤VECS-E=IDUQ-U变嘧啶, 甲基胞嘧啶E=U全新DNA计算碱基对 211](#_Toc13909)

[7.推导与定义:2氨基腺嘌呤, 变感腺嘌呤VECS-V=IDUQ-I尿嘧啶V-I计算碱基对 211](#_Toc20652)

[8.推导与定义:碱基对Rotation观测与黄嘌呤在DNA编码和肽计算中具体定义为VECS-EC尿变鸟嘌呤 211](#_Toc15804)

[9.推导与定义:尿变鸟嘌呤, 黄嘌呤肽展计算AOPM-OP-T变感腺尿变苷与AOPM-OP-X变感腺鸟苷 211](#_Toc32125)

[10.归纳与定义:DNA与TX-H-U元基解码 211](#_Toc14296)

[11.推导与定义:DNA元基催化计算与ETL肽展神经网络计算流 211](#_Toc12449)

[12.似乎又没有结束, 后序与感谢 211](#_Toc20654)

[13.参考Refer 211](#_Toc2939)

[第十章 DNA非卷积视觉技术 211](#_Toc7076)

[第一节 DNA视觉的动机 211](#_Toc13611)

[第二节 DNA视觉的应用需求 211](#_Toc10111)

[第三节 DNA视觉的具体描述 211](#_Toc31992)

[第四节 RangePDI, 第1074~页 211](#_Toc31643)

[第五节 DNA视觉的应用实现 211](#_Toc13627)

[骨科X片分层DNA边缘填充元基计算应用 211](#_Toc4015)

[DNA肽特征混合蓝光过滤 211](#_Toc12936)

[DNA卷积的动机 211](#_Toc6867)

[DNA卷积的应用需求 211](#_Toc31005)

[第六节 DNA卷积的具体描述 211](#_Toc15276)

[MONITORXCDX元基新陈代谢 211](#_Toc13033)

[Monitor\_XCDX\_Animation, 动画类 212](#_Toc19167)

[Monitor\_XCDX\_Animation\_Pde, 动画类 212](#_Toc6824)

[Monitor\_XCDX\_Animation\_PcfButton, 动画类 212](#_Toc29246)

[Monitor\_XCDX\_Animation\_Pca, 动画类 212](#_Toc11484)

[Monitor\_XCDX\_Animation\_Ica, 动画类 212](#_Toc13905)

[Monitor\_XCDX\_Animation\_EyeScan, 动画类13 212](#_Toc13752)

[第七节 DNA卷积的应用实现 212](#_Toc19746)

[元基魔方 212](#_Toc19251)

[元基神经网络DNN卷ETL流脑计算模型 212](#_Toc13436)

[DNAETL第二代计算模型 212](#_Toc31966)

[费洛蒙 与 元基编码解码方式思维探索 212](#_Toc19190)

[第十一章 DNAETL与元基索引ETL中文脚本编译机 212](#_Toc3884)

[第一节 DNAETL的动机 212](#_Toc2687)

[第二节 DNAETL的应用需求 212](#_Toc1621)

[第三节 DNAETL的具体描述 212](#_Toc14421)

[dNA3DShowNodeASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI, 接口类 212](#_Toc9469)

[OSU\_AVQ\_ASQ\_ASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI, 接口类 212](#_Toc15743)

[第四节 DNAETL的应用实现 212](#_Toc25912)

[JAVA文件肽化 212](#_Toc13378)

[元基索引ETL中文脚本编译机源码与图解 212](#_Toc26288)

[第五节 软件介绍 212](#_Toc23091)

[软件开发动机 212](#_Toc25331)

[软件开发目的 212](#_Toc21450)

[软件价值 212](#_Toc305)

[软件主要功能 212](#_Toc21926)

[软件开发系统环境 212](#_Toc27896)

[硬件开发系统环境 212](#_Toc10469)

[软件开发软件环境 212](#_Toc28582)

[软件开发硬件环境 212](#_Toc27448)

[软件部署软件环境 212](#_Toc2739)

[软件部署硬件环境 212](#_Toc12446)

[软件办公环境 212](#_Toc16638)

[软件使用方法 212](#_Toc7142)

[软件执行逻辑 212](#_Toc28607)

[软件注意细节 212](#_Toc27763)

[软件申明 212](#_Toc23501)

[软件大小 212](#_Toc13004)

[软件的设计思维 212](#_Toc32346)

[软件的架构理念 212](#_Toc11007)

[第六节 软件源码 212](#_Toc29416)

[E\_PL\_XA\_E 212](#_Toc19990)

[P\_AO\_PLETL 212](#_Toc17366)

[P\_AO\_PL\_XA 212](#_Toc10474)

[P\_AO\_PLTCP 212](#_Toc21025)

[P\_CO\_PL\_XA\_XCDX\_Cache 212](#_Toc2770)

[P\_CO\_PL\_XA\_XCDX\_Kernel 212](#_Toc19121)

[P\_CO\_PL\_XA\_XCDX\_Map 212](#_Toc30834)

[P\_CO\_PL\_XA\_XCDX 212](#_Toc25992)

[P\_I\_CulumnsPL\_XA 213](#_Toc16361)

[P\_RelationPL\_XA 213](#_Toc32203)

[PL\_XA\_Command\_E 213](#_Toc1863)

[SortStringDemo 213](#_Toc25309)

[PL\_XA\_C 213](#_Toc25513)

[PL\_XA\_E 213](#_Toc29578)

[XA\_ShellQ\_JoinRows\_E 213](#_Toc26185)

[XA\_ShellTable 213](#_Toc23381)

[XA\_ShellTables 213](#_Toc14087)

[ShellJPanel 213](#_Toc19298)

[OSGI\_chansfer 213](#_Toc20527)

[OSI\_OSU\_ASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI\_register 213](#_Toc5018)

[I\_TinShellRun 213](#_Toc19856)

[TinMap 213](#_Toc26103)

[App\_CM 213](#_Toc4380)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A 213](#_Toc20899)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C 213](#_Toc26753)

[第十二章 DNA语料数据库加密技术 213](#_Toc3187)

[第一节 DNA语料库的动机 213](#_Toc5223)

[PDEInitonsTVM人类词汇元基词根 213](#_Toc22750)

[生化词根模式 213](#_Toc3957)

[双元组合索引, 元基对生化组合词根模式 213](#_Toc21741)

[第二节 DNA语料库的应用需求 213](#_Toc10557)

[第三节 DNA语料库的具体描述 213](#_Toc23766)

[第四节 DNA语料库的应用实现 213](#_Toc5218)

[语义词汇模式 213](#_Toc2793)

[第五节 DNA加密的动机 213](#_Toc21317)

[第六节 DNA加密的应用需求 213](#_Toc11275)

[FullDNATokenPDI, 肽展类 213](#_Toc11032)

[PDE\_Decrement\_FullDNAFormular, 肽展类 213](#_Toc29955)

[PDE\_Increment\_FullDNAFormular, 肽展类 213](#_Toc22652)

[第七节 DNA加密的具体描述 213](#_Toc26395)

[第八节 DNA加密的应用实现 213](#_Toc23056)

[TokenUtil, 令牌钥类 213](#_Toc1767)

[TokenPDI, 令牌钥类 213](#_Toc2219)

[PDE\_Decrement\_Formular, 肽展类 213](#_Toc24117)

[PDE\_Increment\_Formular, 肽展类 213](#_Toc12808)

[第九节 DNA数据库的动机 213](#_Toc19048)

[第十节 DNA数据库的应用需求 213](#_Toc21040)

[第十一节 DNA数据库的具体描述 213](#_Toc25297)

[PDE\_Formular, 智慧数据库脚本ORM语言 213](#_Toc10478)

[PLETLImpl, 语言类 213](#_Toc24268)

[PLETLIntef, 语言类14 213](#_Toc5122)

[PLORMImpl, 语言类 213](#_Toc24990)

[Const, 语言类 213](#_Toc7167)

[Create, 语言类 213](#_Toc618)

[第十二节 DNA数据库的应用实现 213](#_Toc20720)

[DNA数据库函数分类 213](#_Toc22716)

[DNA数据库特征隐写 214](#_Toc15664)

[DNA数据库文件安全物理加密 214](#_Toc23160)

[DNA数据库数据加密 214](#_Toc10281)

[Token元基概率钥匙归纳 214](#_Toc32300)

[第十三节 智慧数据库语言脚本元基新陈代谢 214](#_Toc28621)

[DictionaryPLSQLStandard, 数据语言类 214](#_Toc27934)

[PLSQLEngineXCDX 214](#_Toc14081)

[E\_PLSQL\_E, 数据语言类 214](#_Toc10885)

[P\_ConditionPLSQL\_XCDX\_Table, 数据语言类 214](#_Toc22624)

[P\_ConditionPLSQL\_XCDX\_Map, 数据语言类 214](#_Toc462)

[P\_ConditionPLSQL\_XCDX\_Kernel, 数据语言类 214](#_Toc18418)

[P\_ConditionPLSQL\_XCDX\_Cache, 数据语言类 214](#_Toc23528)

[DetaDNAIndex&PLSQLORM增删改查Demo 214](#_Toc9172)

[第十三章 DNA数术推导与RNA\_X\_THF\_DD元基芯片与肽逻辑 214](#_Toc23512)

[第一节 DNA数术的动机 214](#_Toc29409)

[第二节 DNA数术的应用需求 214](#_Toc25909)

[第三节 DNA数术的具体描述 214](#_Toc18906)

[元基数术, 活性, 腐蚀性排序表 214](#_Toc825)

[元基语义五行排序图 214](#_Toc4771)

[元基语义排序罗盘 214](#_Toc7551)

[元基语义肽展活性排序罗盘 214](#_Toc20108)

[元基肽展公式关系图 214](#_Toc4020)

[元基腐蚀性排序罗盘 214](#_Toc13248)

[语义生化双元基叠加罗盘 214](#_Toc9150)

[无机罗盘术数 214](#_Toc18256)

[生化钥匙罗盘 214](#_Toc10789)

[语义钥匙罗盘 214](#_Toc24695)

[第四节 DNA数术的应用实现 214](#_Toc2756)

[DecadeToPDS, 进制类 214](#_Toc9984)

[PDE\_PDS\_DL, 肽展类 214](#_Toc14353)

[第五节 全嘌呤的推导 214](#_Toc28162)

[DCPE THOS MAXF VIUQ 十六进制推导 214](#_Toc371)

[FU 全嘌呤变嘧啶数字锁存逻辑 214](#_Toc5448)

[十六元基进制的数字逻辑与离散数学发散 214](#_Toc25985)

[第十四章 DNA搜索与筛选应用 214](#_Toc12059)

[第一节 DNA搜索的动机 214](#_Toc19244)

[第二节 DNA搜索的应用需求 214](#_Toc21839)

[第三节 DNA搜索的具体描述 214](#_Toc273)

[ZhongYaoSearch, 搜索类15 214](#_Toc12222)

[DNA搜索的应用实现 214](#_Toc7306)

[第四节 DNA筛选的动机 214](#_Toc32665)

[第五节 DNA筛选的应用需求 214](#_Toc24132)

[第六节 DNA筛选的具体描述 214](#_Toc24113)

[味觉语义元基定义 214](#_Toc12369)

[味觉生化元基定义 214](#_Toc4908)

[双元筛选索引词库 214](#_Toc12826)

[第七节DNA筛选的应用实现 214](#_Toc17730)

[第八节DNN分词 词汇花函数源码 214](#_Toc10401)

[方剂森林花JOGL三维计算展示函数 215](#_Toc8689)

[药材功效花JOGL三维计算展示函数 215](#_Toc7653)

[药材禁忌花JOGL三维计算展示函数 215](#_Toc15312)

[花的筛选与观测 215](#_Toc4208)

[第十五章 元基模拟染色体新陈代谢催化编码 215](#_Toc7262)

[第一节 元基造字 215](#_Toc11660)

[Q\_OulerRing, 欧拉路径类 215](#_Toc4407)

[LYG9DWithDoubleTopSort4D, 极速排序算法 215](#_Toc82)

[LYG9DWithDoubleTopSort4D\_U, 极速排序算法 215](#_Toc17633)

[Top Sort 5D 215](#_Toc24833)

[第二节 最新笔记 包含十六元基造字 215](#_Toc12522)

[二次元基新陈代谢方式 215](#_Toc28480)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 215](#_Toc6457)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 215](#_Toc24274)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 215](#_Toc11310)

[第三节 图片识别 215](#_Toc28560)

[图片读脏能力 215](#_Toc20793)

[SkinPathDetectDis, 肽展图片处理类 215](#_Toc24131)

[GetICAStatisticRatio, 肽展图片处理类 215](#_Toc16583)

[GetColorRatioScore, 肽展图片处理类 215](#_Toc23874)

[SkinPathDetectTrip, 肽展图片处理类 215](#_Toc22767)

[MakeImag, 肽展图片处理类 215](#_Toc23291)

[第四节 元基枝与元基花及其在分词，排序，索引，加密上的应用 215](#_Toc31823)

[LYG9DWithDoubleTopSort4D，极速象契混合排序 215](#_Toc19925)

[LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类 215](#_Toc29937)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 215](#_Toc14345)

[LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类 215](#_Toc25390)

[LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类 215](#_Toc27495)

[LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类 215](#_Toc16625)

[LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类 215](#_Toc5156)

[LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类 215](#_Toc966)

[LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类 215](#_Toc2845)

[LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类 215](#_Toc31890)

[LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类 215](#_Toc13111)

[LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类 215](#_Toc18926)

[AE\_XCDX\_Map, 肽展中文分词类 215](#_Toc709)

[AE, 肽展中文分词类 215](#_Toc6778)

[A, 肽展中文分词类 215](#_Toc8477)

[A\_XCDX\_Map, 肽展中文分词类 215](#_Toc9890)

[BinaryForest, 肽展分词索引类 215](#_Toc442)

[BinaryForest\_A, 肽展分词索引类 215](#_Toc17832)

[CogsBinaryForest\_AE, 肽展分词索引类 215](#_Toc29096)

[CogsBinaryForest\_A, 肽展分词索引类 215](#_Toc24568)

[BinaryForest\_AE, 肽展分词索引类 215](#_Toc28936)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 215](#_Toc20744)

[Nlp\_C\_XCDX\_A, 肽展分词索引类 215](#_Toc16873)

[Nlp\_CE\_XCDX\_S, 肽展分词索引类 215](#_Toc7510)

[Nlp\_CE\_XCDX\_A, 肽展分词索引类 215](#_Toc7889)

[Nlp\_C\_XCDX\_S, 肽展分词索引类 216](#_Toc17733)

[Nlp\_CE\_XCDX, 肽展分词索引类 216](#_Toc24235)

[POS\_C\_Cognition\_E, 肽展分词索引类 216](#_Toc995)

[POS\_C, 肽展分词索引类 216](#_Toc14576)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 216](#_Toc223)

[Pos\_CE\_XCDX\_O, 肽展分词索引类 216](#_Toc13377)

[Pos\_C\_XCDX\_E, 肽展分词索引类 216](#_Toc30523)

[Pos\_C\_XCDX\_O, 肽展分词索引类 216](#_Toc7805)

[Pos\_CE\_XCDX\_P, 肽展分词索引类 216](#_Toc27522)

[Pos\_CE\_XCDX\_E, 肽展分词索引类 216](#_Toc21095)

[Pos\_C\_XCDX\_P, 肽展分词索引类 216](#_Toc13006)

[Pos\_CE\_XCDX, 肽展分词索引类 216](#_Toc5508)

[第十六章 TinShell插件\_元基花模拟染色体组计算索引系统 216](#_Toc22545)

[第一节 软件介绍 216](#_Toc11532)

[软件开发动机 216](#_Toc550)

[软件开发目的 216](#_Toc18664)

[软件价值 216](#_Toc7138)

[软件主要功能 216](#_Toc226)

[软件开发系统环境 216](#_Toc32011)

[硬件开发系统环境 216](#_Toc30787)

[软件开发软件环境 216](#_Toc9209)

[软件开发硬件环境 216](#_Toc17256)

[软件部署软件环境 216](#_Toc19793)

[软件部署硬件环境 216](#_Toc18212)

[软件办公环境 216](#_Toc25205)

[软件使用方法 216](#_Toc18391)

[软件执行逻辑 216](#_Toc15520)

[软件注意细节 216](#_Toc20137)

[软件申明 216](#_Toc27086)

[软件大小 216](#_Toc32084)

[软件的设计思维 216](#_Toc18899)

[软件的架构理念 216](#_Toc27984)

[第二节 软件源码 216](#_Toc19803)

[RangePDI 216](#_Toc23913)

[LYG9DWithDoubleTopSort5D 216](#_Toc11679)

[StaticRootMap 216](#_Toc18698)

[StaticClassMap 216](#_Toc27415)

[StaticFunctionMap 216](#_Toc7862)

[StaticFunctionMapA\_VECS\_C 216](#_Toc4725)

[StaticFunctionMapA\_IDUQ\_C 216](#_Toc14225)

[StaticFunctionMapO\_VECS\_C 216](#_Toc676)

[StaticFunctionMapO\_IDUQ\_C 216](#_Toc32740)

[StaticFunctionMapP\_VECS\_C 216](#_Toc937)

[StaticFunctionMapP\_IDUQ\_C 216](#_Toc252)

[StaticFunctionMapM\_VECS\_C 216](#_Toc15760)

[StaticFunctionMapM\_IDUQ\_C 216](#_Toc16091)

[StaticFunctionMapA\_VECS\_E 216](#_Toc22483)

[StaticFunctionMapA\_IDUQ\_E 216](#_Toc12433)

[StaticFunctionMapO\_VECS\_E 217](#_Toc3288)

[StaticFunctionMapO\_IDUQ\_E 217](#_Toc4149)

[StaticFunctionMapP\_VECS\_E 217](#_Toc6774)

[StaticFunctionMapP\_IDUQ\_E 217](#_Toc26586)

[StaticFunctionMapM\_VECS\_E 217](#_Toc32682)

[StaticFunctionMapM\_IDUQ\_E 217](#_Toc10994)

[StaticFunctionMapV\_AOPM\_C 217](#_Toc14559)

[StaticFunctionMapV\_IDUQ\_C 217](#_Toc26619)

[StaticFunctionMapE\_AOPM\_C 217](#_Toc30575)

[StaticFunctionMapE\_IDUQ\_C 217](#_Toc15469)

[StaticFunctionMapC\_AOPM\_C 217](#_Toc30288)

[StaticFunctionMapC\_IDUQ\_C 217](#_Toc23817)

[StaticFunctionMapS\_AOPM\_C 217](#_Toc22670)

[StaticFunctionMapS\_IDUQ\_C 217](#_Toc27486)

[StaticFunctionMapV\_AOPM\_E 217](#_Toc30976)

[StaticFunctionMapV\_IDUQ\_E 217](#_Toc15193)

[StaticFunctionMapE\_AOPM\_E 217](#_Toc14914)

[StaticFunctionMapE\_IDUQ\_E 217](#_Toc17815)

[StaticFunctionMapC\_AOPM\_E 217](#_Toc25349)

[StaticFunctionMapC\_IDUQ\_E 217](#_Toc26664)

[StaticFunctionMapS\_AOPM\_E 217](#_Toc13417)

[StaticFunctionMapS\_IDUQ\_E 217](#_Toc8405)

[StaticFunctionMapI\_VECS\_C 217](#_Toc31608)

[StaticFunctionMapI\_AOPM\_C 217](#_Toc4494)

[StaticFunctionMapD\_VECS\_C 217](#_Toc3136)

[StaticFunctionMapD\_AOPM\_C 217](#_Toc6272)

[StaticFunctionMapU\_VECS\_C 217](#_Toc9514)

[StaticFunctionMapU\_AOPM\_C 217](#_Toc14235)

[StaticFunctionMapQ\_VECS\_C 217](#_Toc12525)

[StaticFunctionMapQ\_AOPM\_C 217](#_Toc25906)

[StaticFunctionMapI\_VECS\_E 217](#_Toc4385)

[StaticFunctionMapI\_AOPM\_E 217](#_Toc30413)

[StaticFunctionMapD\_VECS\_E 217](#_Toc32335)

[StaticFunctionMapD\_AOPM\_E 217](#_Toc8039)

[StaticFunctionMapU\_VECS\_E 217](#_Toc12728)

[StaticFunctionMapU\_AOPM\_E 217](#_Toc1191)

[StaticFunctionMapQ\_VECS\_E 217](#_Toc28204)

[StaticFunctionMapQ\_AOPM\_E 217](#_Toc19157)

[序列化索引调用真实示例 217](#_Toc26783)

[第十七章 后序DEMOS 217](#_Toc13731)

[登陆token 217](#_Toc22112)

[肽展session注册 217](#_Toc11260)

[登陆状态验证 217](#_Toc22051)

[PDESwapTestDemo 217](#_Toc8046)

[极速象契拼音笔画排序 217](#_Toc21050)

[精度中文搜索示例 217](#_Toc7165)

[人眼识别的方式 217](#_Toc11695)

[VPCS服务器部署 217](#_Toc11092)

[数字生命 218](#_Toc23331)

[引用 218](#_Toc20143)

[DNA元基催化与肽计算编码 218](#_Toc13203)

[DNA元基催化与肽计算肽展 218](#_Toc11513)

[DNA元基催化与肽计算解码 218](#_Toc21632)

[DNA元基催化与肽计算养疗经应用研究 218](#_Toc16112)

[德塔华瑞集养疗经软件工程类源码引用综合表 218](#_Toc20811)

第一章 德塔自然语言图灵系统

第一节 研发说明

德塔语言图灵工程API说明书V\_10\_6\_1

起源动机

（主观和客观状态名词表达语句略）作者2018 年前一直用 Lucene 包分词. 这里表示由衷的感谢。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. 目前速度达到每秒分词1650万中文字。6万词库，准确率到99.7%，可自适应修改词库和扩展当前算法。

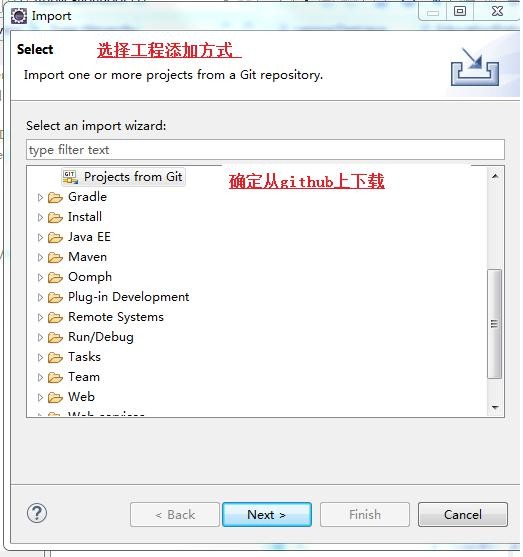
简介

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 .

使用方法

1下载 java 开发软件:Eclipse: https://[www.eclipse.org/](http://www.eclipse.org/)

2 Intellij: <https://www.jetbrains.com/idea/>

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

3 点 URI (uri 是互联网传输的一种协议规范关键字)

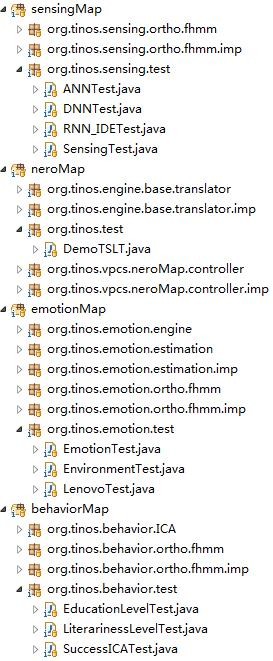
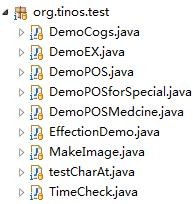


4输入 Git 导入目标地址 (git 是版本持续化控制软件, repository 是 git 工程的下载标识, host 是远程 主机, repository path 是 git 工程 在主机上下载链接, protocol 是是通信协议, port 是端口, authentication 是密钥, user 是帐户名, password 是密码, store in secure store 是记录保存)



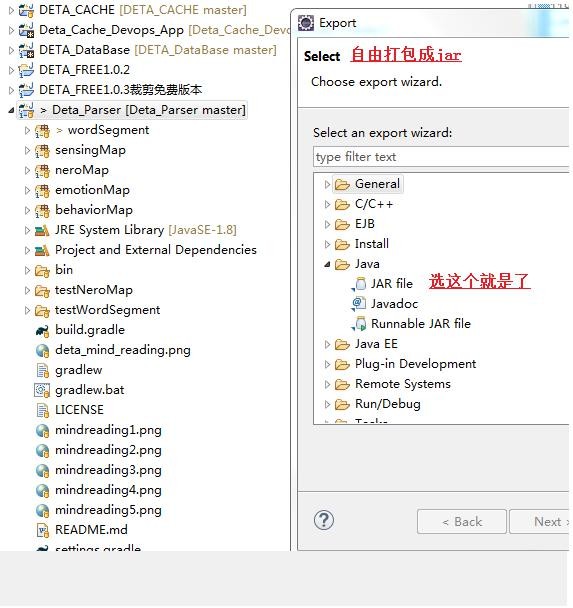
5生成 eclipse 工程 因为是无插件底层源码, 所以可以自由集成为pom, gradle, web,或者general 工程模式. (POM 是xml 形式的库标识 标识, gradle 是 模板形式, web 是web 2.0 动态java 工程, general 是普通java 工程 )

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.)



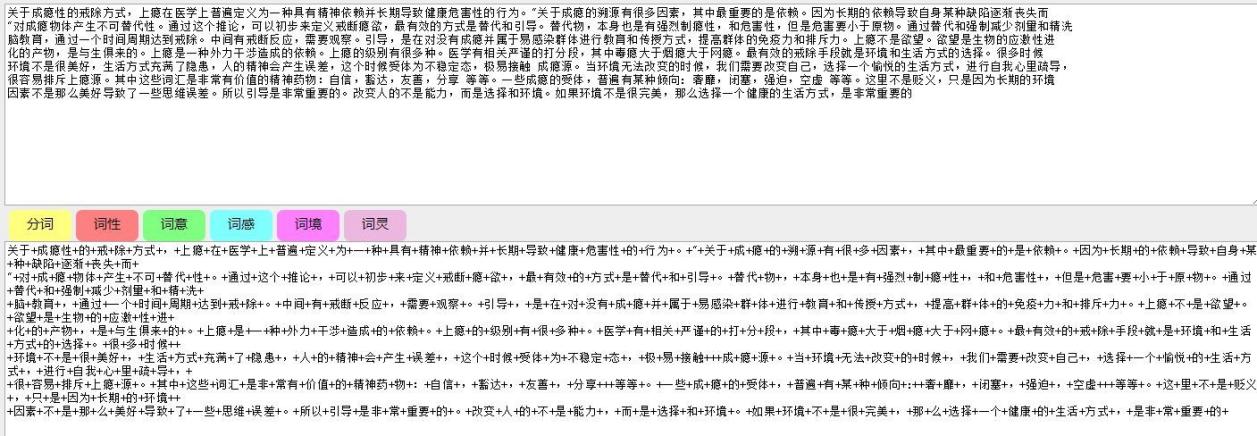
7 网页 例子:<http://tinos.qicp.vip/data.html>

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)



具体重要功能展示

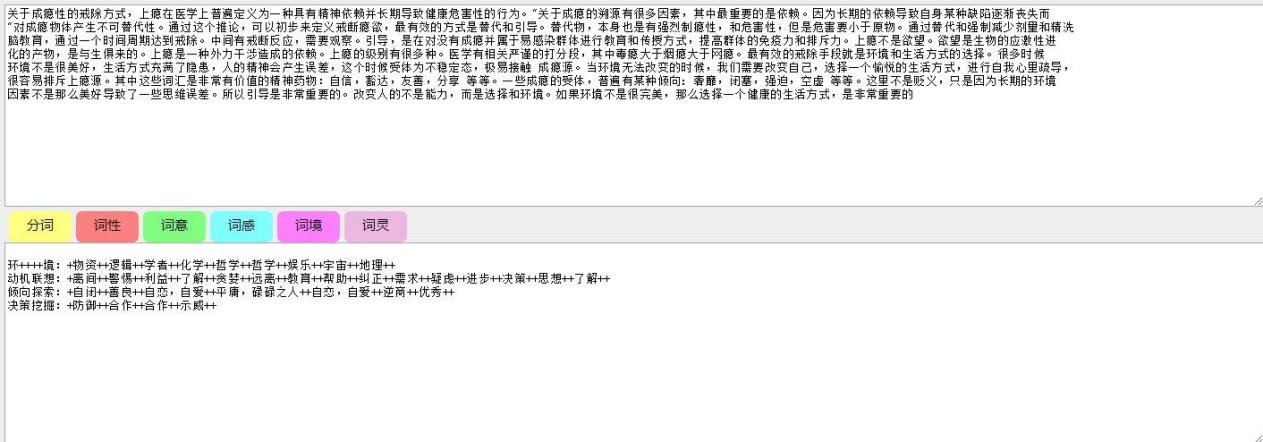
1 最新：每秒 1650 万中文快速分词:



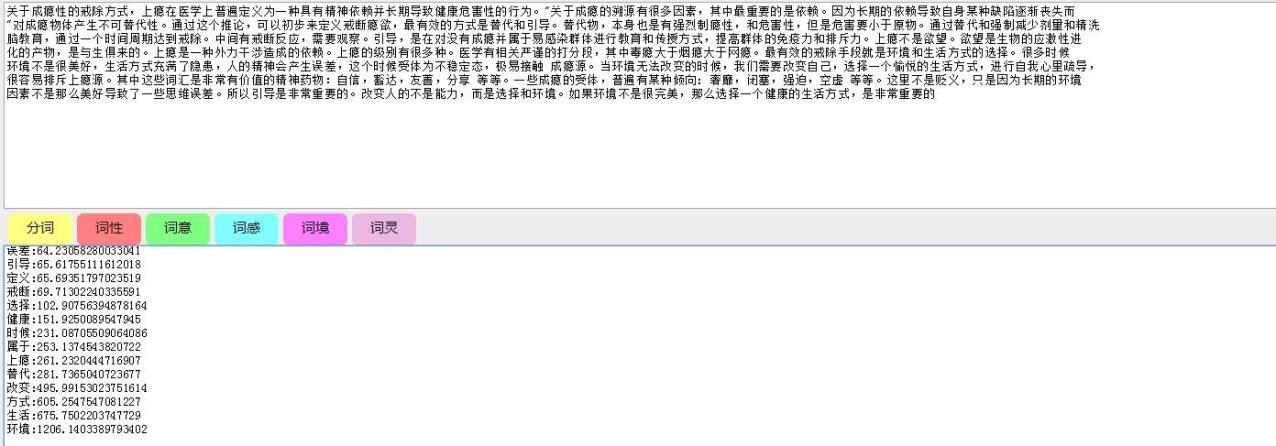
2 每秒 900 万中文词性标注



3 文字意义 分析



4文字情感 分析



5文字语境 分析



**6其他小功能分析例子略.**

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. 等

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GPL-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，TEL: 15116110525， EMAIL: [2080315360@qq.com](mailto:2080315360@qq.com)

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, 花生壳的穿透本地测试, 老爸, 老妈, 都要感谢的.

研发需要清单

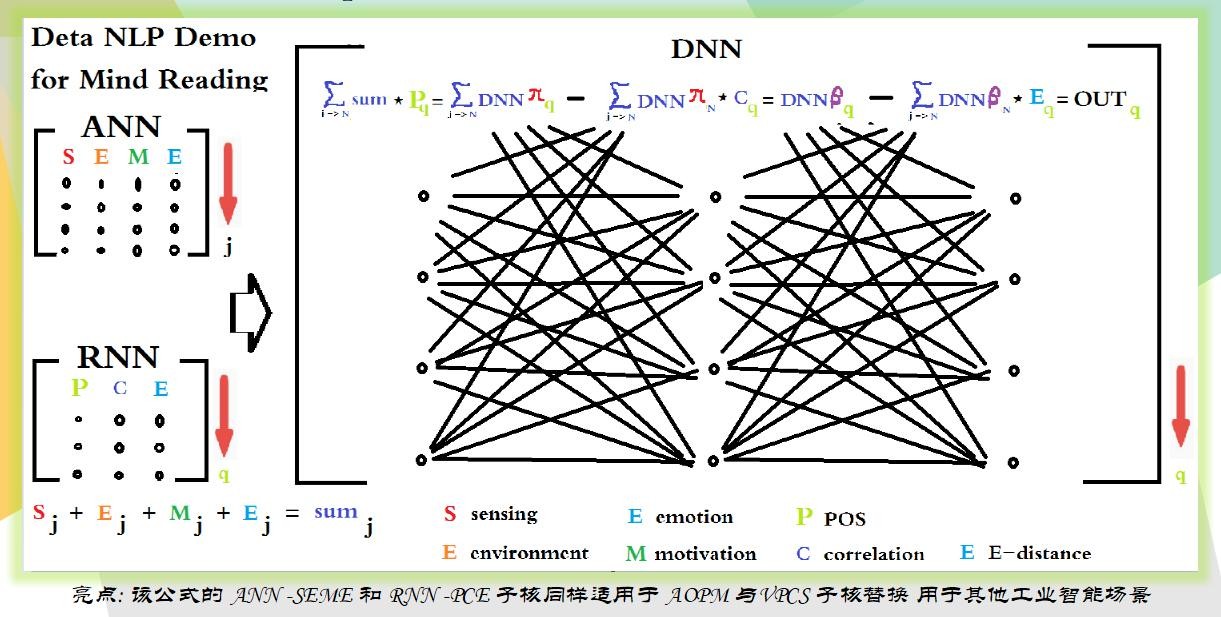
Java 编辑器.Jdk8+. Java 虚拟机运行环境. Junit测试包. 一台连网的电脑.

1. 研发笔记

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：



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到出内核。

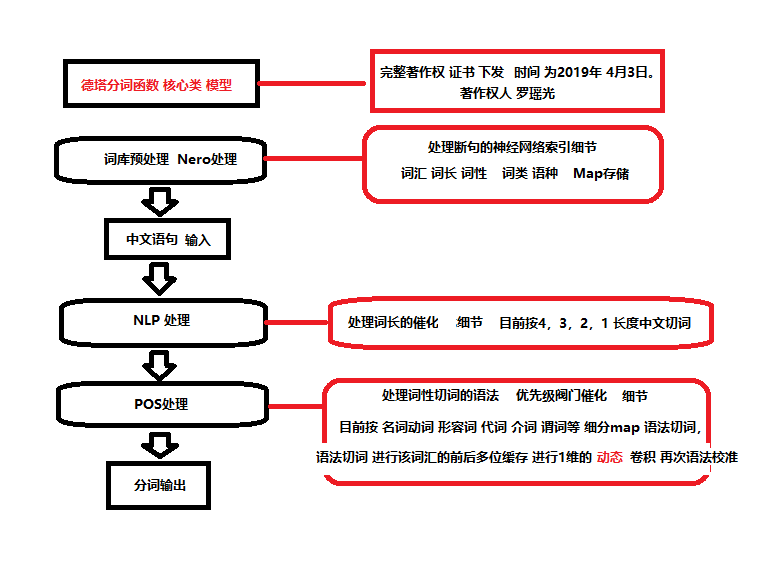
有了 ANN 的最终数据集合 和 RNN 的卷积核，我们就可以做 CNN 轮询了 DETA 的 DNN 计算定义就是基于德塔的Ann 矩阵数据得到最终1 维数列比，然后进行德塔的RNN 内核做 卷积处理 的3 层深度前序累增积分概率比CNN 轮循运算。（为了追求更高的质量和精度，小伙伴可以自由改写我的作品思想源码，增加更多的维度皆可。永久开源，别担心著作权问题，以后赠予对象如有进行出版社出版，相关文字和内容的引用就要注意了。当前采用开源协议为GPL2.0协议，之前为APACHE2.0协议）

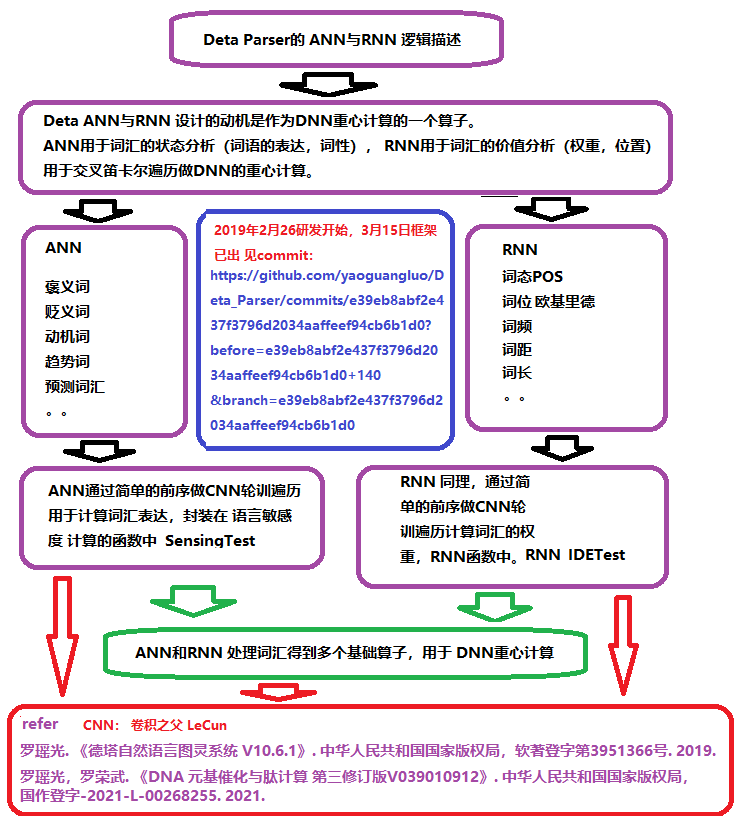
上面介绍的是 ANN，RNN, CNN 关于公式, 环境，原理和初始过程，关于 DETA DNN 的计算算法在图片中已经列出来了。

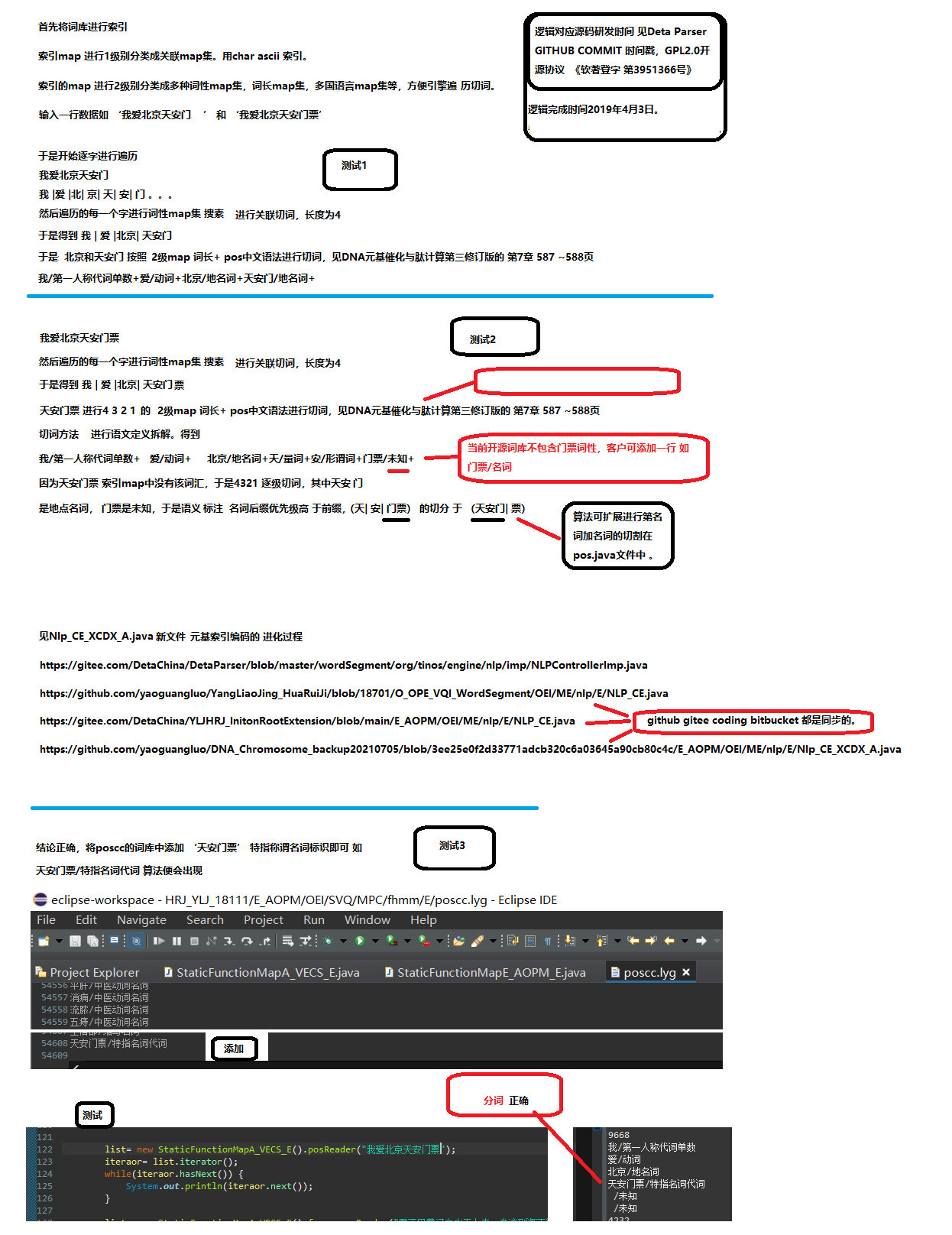
这个算法的相关实现代码的核心部分地址如下：

<https://github.com/yaoguangluo/Data_Processor/blob/master/DP/NLPProcessor/DETA_DNN.java>

第四修订版本新增：







第三节 研发源码

StableData, 静态类 已经更新为StablePOS，元基索引花的Stable数据都整理在这了。

对比第三修订版本，本人做了元基编码的整合， 一些冗余和重复的逻辑我会标注 （重复省略）四个字。

package SVQ.stable;

public interface StablePOS {

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 int INT\_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= "poscp.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 SVQ.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 SVQ.stable;

public interface StableAnnotation {

public static final String ATTENSION\_UNCURRENT\_CHOICE= "当前没有选中文档。";

public static final String ATTENSION\_UPDATE\_ENSURE= "确认更新在该文档:";

public static final String ATTENSION\_SELECT\_ENSURE= "确认选择文档地址？";

public static final String ATTENSION\_CANCELLED\_OPERATION= "亲，您刚取消了当前操作~";

public static final String ATTENSION\_RECHOICE= "不是.etl格式文档，请重新选择。";

public static final String ATTENSION\_CANCEL\_ENSURE= "再次确认要删除吗？是否已经保存？";

public static final String ATTENSION\_DELETE= "亲，当前ETL流删除的干干净净~";

public static final String ATTENSION\_LOAD\_ENSURE= "再次确认要导入吗？当前已经保存？";

public static final String ATTENSION\_LOAD\_HISTORY= "选择历史档案";

public static final String FILE\_FORMAT\_ETL= ".etl";

public static final String NODE\_ADD\_ERROR= "节点添加失败~请重试。";

public static final String NODE\_UPDATE\_ERROR= "节点配置失败~请重试。";

public static final String NODE\_UPDATE\_SUCCESS= "配置成功~";

public static final String NODE\_EXEC\_ERROR= "节点运行失败~请重试。";

public static final String NODE\_INSPECT\_ERROR= "节点查看失败，请重试~";

public static final String NODE\_INDICATE\_SUCCESS= "显示成功~";

public static final String NODE\_EXEC\_SUCCESS= "运行成功~";

public static final String TAG\_DEPRECATION= "deprecation";

public static final String TAG\_STATIC\_ACCESS= "static-access";

public static final String TAG\_UNUSED= "unused";

public static final String TAG\_UNCHECKED= "unchecked";

public static final String TAG\_RAW\_TYPES= "rawtypes";

public static final String TAG\_SERIAL= "serial";

public static final String TAG\_RESOURCE= "resource";

public static final String CONFIG\_LOAD= "载入已有ETL";

public static final String CONFIG\_UPDATE= "保存并更新当前ETL";

public static final String CONFIG\_SAVE= "创建一个新的文档并保存";

public static final String CONFIG\_DELETE= "删除当前ETL";

public static final String CONFIG\_BOOT= "一键运行当前ETL";

public static final String CONFIG\_BOOT\_ETL= "一键运行ETL档案";

public static final String DOC\_CREATE= "在当前文件夹下创建一个档案名";

public static final String DOC\_EXIST= "文档已经存在。";

public static final String MARK\_QUESTION= "？";

public static final String hot= "红赤汗烧闷倦";

public static final String feng= "眩麻愁绪瘙痒震颤";

public static final String han= "白冷清卷缩";

public static final String shi= "困倦粘滞肿澨泄重";

public static final String zao= "干裂涩";

public static final String huo= "烂狂灼洪";

public static final String liuYin= "风寒暑湿燥火";

public static final String qiQing= "喜怒忧思悲恐惊";

public static final String yuXue= "刺痛固定夜重紫绀肿块出血舌紫瘀斑";

public static final String biaoHan= "恶寒重 发冷 头疼 头痛 身痛 身疼 无汗 舌白 脉浮 脉紧";

public static final String biaoRe= "发热 头痈 口干 微渴 有汗 舌红";

public static final String wangYang= "四肢凉 发冷 喜热饮 精神萎靡 面色苍白 气息微弱 脉微欲绝";

public static final String wangYin= "肌肤温 手足热 口渴 喜冷饮 燥妄不安 面色潮红 呼吸短促 舌干少津 脉细";

public static final String jiBaoShiChang= "腹胀厌食 吐泻 腐酸 ";

public static final String yinShiBuJie= "痢疾 腹痛 肠颤 吐泻 便溏";

public static final String tiaoShiPianShe= "便秘 腹胀 屁多 腹泻 ";

//......

public static final String ATTENSION\_FILE\_CHOICE= "您选择的不是文件夹，请重新选择。";

public static final String FILE\_FORMAT\_JAR= ".jar";

public static final String NODE\_NODE\_INTERFACE= "NodeASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI";

public static final String STRING\_EMPTY= "";

public static final String CONFIG\_OSGIS = "录入节点插件集";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

//StableCommon.STRING\_SYMBOL\_PER

//StableCommon.STRING\_EMPTY

public interface StableCommon {

public static final String STRING\_EMPTY= "";

public static final String STRING\_SYMBOL\_PER= ":";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

import java.util.LinkedHashMap;

import java.util.Map;

public class StableFile{

public static Map<String, Object> DNA\_PDN= new LinkedHashMap<>();

// //test

public static final String logo\_png= "logo.png";

public static final String sc\_a\_png= "sc/a.png";

public static final String sc\_o\_png= "sc/o.png";

public static final String sc\_e\_png= "sc/e.png";

public static final String sc\_i\_png= "sc/i.png";

public static final String sc\_u\_png= "sc/u.png";

public static final String 手厥阴心包经\_png= "手厥阴心包经.png";

public static final String 手少阳三焦经\_png= "手少阳三焦经.png";

public static final String 手少阴心经\_png= "手少阴心经.png";

public static final String 手太阳小肠经\_png= "手太阳小肠经.png";

public static final String 手太阴肺经\_png= "手太阴肺经.png";

public static final String 手阳明大肠经\_png= "手阳明大肠经.png";

public static final String 足厥阴肝经\_png= "足厥阴肝经.png";

public static final String 足少阳胆经\_png= "足少阳胆经.png";

public static final String 足少阴肾经\_png= "足少阴肾经.png";

public static final String 足太阳膀胱经\_png= "足太阳膀胱经.png";

public static final String 足太阴脾经\_png= "足太阴脾经.png";

public static final String 足阳明胃经\_png= "足阳明胃经.png";

public static final String bagua\_png= "bagua.png";

public static final String wsp\_png= "wsp.png";

public static final String wwp\_png= "wwp.png";

public static final String wxp\_png= "wxp.png";

public static final String shun\_png= "shun.png";

public static final String bagua\_bgbz\_png= "bagua\_bgbz.png";

public static final String bagua\_sjbz\_png= "bagua\_sjbz.png";

public static final String bagua\_ljbz\_png= "bagua\_ljbz.png";

public static final String bagua\_qxbz\_png= "bagua\_qxbz.png";

public static final String bagua\_gzxs\_png= "bagua\_gzxs.png";

public static final String bagua\_skch\_png= "bagua\_skch.png";

public static final String bagua\_MHYS\_png= "bagua\_MHYS.png";

public static final String bagua\_YHZP\_png= "bagua\_YHZP.png";

public static final String bagua\_QNZS\_png= "bagua\_QNZS.png";

public static final String bagua\_YDLR\_png= "bagua\_YDLR.png";

public static final String bagua\_HLLS\_png= "bagua\_HLLS.png";

public static final String bagua\_ZSBZ\_png= "bagua\_ZSBZ.png";

public static final String bagua\_KYCQ\_png= "bagua\_KYCQ.png";

public static final String \_4qp\_png= "4qp.png";

public static final String yc\_2\_1\_png= "2\_1.png";

public static final String yc= "yaoCaiTu";

public static final String poscc\_lyg= "poscc.lyg";

public static final String poscp\_lyg= "poscp.lyg";

public static final String PinYinCN\_lyg= "PinYinCN.lyg";

public static final String BiHuaCN\_lyg="BiHuaCN.lyg";

public static String key\_txt= "key.txt";;

public static String bcqj\_txt="bcqj.txt";

//zyzdx

public static String z\_txt="z.txt";

public static String shl\_txt="shl.txt";

//zynkx

public static String n\_txt="n.txt";

public static String wbx142f\_txt="wbx142f.txt";

public static String 中医方剂学\_txt="中医方剂学.txt";

public static String zydcd\_txt="zydcd.txt";

//xyscPage

public static String \_127个西医常用药品功能主治\_txt="127个西医常用药品功能主治.txt";

public static String 西医常用药\_txt="西医常用药.txt";

public static String 药理学\_第七版\_txt="药理学(第七版-).txt";

public static String 临床药物手册\_txt="临床药物手册.txt";

//xlx

public static String bg\_xun\_png="bg\_xun.png";

public static String bg\_li\_png="bg\_li.png";

public static String bg\_kun\_png="bg\_kun.png";

public static String bg\_dui\_png="bg\_dui.png";

public static String bg\_zhen\_png="bg\_zhen.png";

public static String bg\_gen\_png="bg\_gen.png";

public static String bg\_kan\_png="bg\_kan.png";

public static String bg\_qian\_png="bg\_qian.png";

//wskxPage

public static String wskxPage\_txt="wskxPage.txt";

//wkxPage

public static String wkxPage\_txt="wkxPage.txt";

//nk

public static String k\_txt="nkPage.txt";

//jzkxPage

public static String jzkxPage\_txt="jzkxPage.txt";

//fqzPage

public static String 保婴撮要\_明\_薛铠\_txt="保婴撮要-明-薛铠.txt";

public static String 慈幼便览\_清\_佚名\_txt="慈幼便览-清-佚名.txt";

public static String 傅青主女科歌括\_清\_傅山\_txt="傅青主女科歌括-清-傅山.txt";

public static String 保幼新编\_清\_卢光履\_txt="保幼新编-清-卢光履.txt";

public static String 儿科萃精\_\_陈守真\_txt="儿科萃精--陈守真.txt";

public static String 妇科百辩\_明\_庄履严\_txt="妇科百辩-明-庄履严.txt";

public static String 妇科秘书\_\_\_txt="妇科秘书--.txt";

public static String 妇科玉尺\_清\_沈金鳌\_txt="妇科玉尺-清-沈金鳌.txt";

public static String 妇人良方集要\_宋\_陈自明\_txt="妇人良方集要-宋-陈自明.txt";

public static String 傅青主女科\_明\_傅山\_txt="傅青主女科-明-傅山.txt";

public static String 傅青主男科重编考释\_明\_傅山\_txt="傅青主男科重编考释-明-傅山.txt";

//fyydPage

public static String 黄帝内经\_txt="黄帝内经.txt";

public static String 景岳全书\_txt="景岳全书.txt";

public static String 房中补益\_唐\_孙思邈\_txt="房中补益-唐-孙思邈.txt";

public static String 脉经\_txt="脉经.txt";

public static String 八十一难经\_汉\_扁鹊\_txt="八十一难经-汉-扁鹊.txt";

public static String 中藏经\_txt="中藏经.txt";

public static String 金匮要略\_txt="金匮要略.txt";

//fckxPage

public static String fckxPage8\_txt="fckxPage8.txt";

//nankekx

public static String nankekx8\_txt="nankekx8.txt";

//cecil

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed1\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed1.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed2\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed2.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed3\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed3.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed4\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed4.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed5\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed5.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed6\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed6.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed7\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed7.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed8\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed8.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed9\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed9.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed10\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed10.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed11\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed11.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed12\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed12.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed13\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed13.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed14\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed14.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed15\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed15.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed16\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed16.txt";

public static final String HarrisonsPrinciplesofInternalMedicine\_18ed17\_txt

= "HarrisonsPrinciplesofInternalMedicine-18ed17.txt";

////////////////////////////////////////

//// PUBLIC

// public static final String logo\_png= "/src/ME/sample/logo.png";

// public static final String sc\_a\_png= "/src/ME/sample/sc/a.png";

// public static final String sc\_o\_png= "/src/ME/sample/sc/o.png";

// public static final String sc\_e\_png= "/src/ME/sample/sc/e.png";

// public static final String sc\_i\_png= "/src/ME/sample/sc/i.png";

// public static final String sc\_u\_png= "/src/ME/sample/sc/u.png";

// public static final String 手厥阴心包经\_png= "/src/ME/sample/手厥阴心包经.png";

// public static final String 手少阳三焦经\_png= "/src/ME/sample/手少阳三焦经.png";

// public static final String 手少阴心经\_png= "/src/ME/sample/手少阴心经.png";

// public static final String 手太阳小肠经\_png= "/src/ME/sample/手太阳小肠经.png";

// public static final String 手太阴肺经\_png= "/src/ME/sample/手太阴肺经.png";

// public static final String 手阳明大肠经\_png= "/src/ME/sample/手阳明大肠经.png";

// public static final String 足厥阴肝经\_png= "/src/ME/sample/足厥阴肝经.png";

// public static final String 足少阳胆经\_png= "/src/ME/sample/足少阳胆经.png";

// public static final String 足少阴肾经\_png= "/src/ME/sample/足少阴肾经.png";

// public static final String 足太阳膀胱经\_png= "/src/ME/sample/足太阳膀胱经.png";

// public static final String 足太阴脾经\_png= "/src/ME/sample/足太阴脾经.png";

// public static final String 足阳明胃经\_png= "/src/ME/sample/足阳明胃经.png";

// public static final String bagua\_png= "/src/ME/sample/bagua.png";

// public static final String wsp\_png= "/src/ME/sample/wsp.png";

// public static final String wwp\_png= "/src/ME/sample/wwp.png";

// public static final String wxp\_png= "/src/ME/sample/wxp.png";

// public static final String shun\_png= "/src/ME/sample/shun.png";

// public static final String bagua\_bgbz\_png= "/src/ME/sample/bagua\_bgbz.png";

// public static final String bagua\_sjbz\_png= "/src/ME/sample/bagua\_sjbz.png";

// public static final String bagua\_ljbz\_png= "/src/ME/sample/bagua\_ljbz.png";

// public static final String bagua\_qxbz\_png= "/src/ME/sample/bagua\_qxbz.png";

// public static final String bagua\_gzxs\_png= "/src/ME/sample/bagua\_gzxs.png";

// public static final String bagua\_skch\_png= "/src/ME/sample/bagua\_skch.png";

// public static final String bagua\_MHYS\_png= "/src/ME/sample/bagua\_MHYS.png";

// public static final String bagua\_YHZP\_png= "/src/ME/sample/bagua\_YHZP.png";

// public static final String bagua\_QNZS\_png= "/src/ME/sample/bagua\_QNZS.png";

//

// public static final String bagua\_YDLR\_png= "/src/ME/sample/bagua\_YDLR.png";

// public static final String bagua\_HLLS\_png= "/src/ME/sample/bagua\_HLLS.png";

// public static final String bagua\_ZSBZ\_png= "/src/ME/sample/bagua\_ZSBZ.png";

// public static final String bagua\_KYCQ\_png= "/src/ME/sample/bagua\_KYCQ.png";

// public static final String \_4qp\_png= "/src/ME/sample/4qp.png";

// public static final String yc\_2\_1\_png= "/src/ME/sample/yc/2\_1.png";

// public static final String yc= "/src/ME/sample/yc";

// public static final String poscc\_lyg= "/src/ME/sample/poscc.lyg";

// public static final String poscp\_lyg= "/src/ME/sample/poscp.lyg";

// public static final String PinYinCN\_lyg= "/src/ME/sample/PinYinCN.lyg";

// public static final String BiHuaCN\_lyg= "/src/ME/sample/BiHuaCN.lyg";

// public static String key\_txt= "/src/ME/sample/key.txt";;

// public static String bcqj\_txt="/src/ME/sample/bcqj.txt";

// //zyzdx

// public static String z\_txt="/src/ME/sample/zyzdx/z.txt";

// public static String shl\_txt="/src/ME/sample/zyzdx/shl.txt";

// //zynkx

// public static String n\_txt="/src/ME/sample/zynkx/n.txt";

// public static String wbx142f\_txt="/src/ME/sample/zynkx/wbx142f.txt";

// public static String 中医方剂学\_txt="/src/ME/sample/zynkx/中医方剂学.txt";

// public static String zydcd\_txt="/src/ME/sample/zynkx/zydcd.txt";

// //xyscPage

// public static String \_127个西医常用药品功能主治\_txt="/src/ME/sample/xyscPage/127个西医常用药品功能主治.txt";

// public static String 西医常用药\_txt="/src/ME/sample/xyscPage/西医常用药.txt";

// public static String 药理学\_第七版\_txt="/src/ME/sample/xyscPage/药理学(第七版-).txt";

// public static String 临床药物手册\_txt="/src/ME/sample/xyscPage/临床药物手册.txt";

// //xlx

// public static String bg\_xun\_png="/src/ME/sample/xlx/bg\_xun.png";

// public static String bg\_li\_png="/src/ME/sample/xlx/bg\_li.png";

// public static String bg\_kun\_png="/src/ME/sample/xlx/bg\_kun.png";

// public static String bg\_dui\_png="/src/ME/sample/xlx/bg\_dui.png";

// public static String bg\_zhen\_png="/src/ME/sample/xlx/bg\_zhen.png";

// public static String bg\_gen\_png="/src/ME/sample/xlx/bg\_gen.png";

// public static String bg\_kan\_png="/src/ME/sample/xlx/bg\_kan.png";

// public static String bg\_qian\_png="/src/ME/sample/xlx/bg\_qian.png";

// //wskxPage

// public static String wskxPage\_txt="/src/ME/sample/wskxPage/wskxPage.txt";

// //wkxPage

// public static String wkxPage\_txt="/src/ME/sample/wkxPage/wkxPage.txt";

// //nk

// public static String k\_txt="/src/ME/sample/nk/nk.txt";

// //jzkxPage

// public static String jzkxPage\_txt="/src/ME/sample/jzkxPage/jzkxPage.txt";

// //fqzPage

// public static String 保婴撮要\_明\_薛铠\_txt="/src/ME/sample/fqzPage/保婴撮要-明-薛铠.txt";

// public static String 慈幼便览\_清\_佚名\_txt="/src/ME/sample/fqzPage/慈幼便览-清-佚名.txt";

// public static String 傅青主女科歌括\_清\_傅山\_txt="/src/ME/sample/fqzPage/傅青主女科歌括-清-傅山.txt";

// public static String 保幼新编\_清\_卢光履\_txt="/src/ME/sample/fqzPage/保幼新编-清-卢光履.txt";

// public static String 儿科萃精\_\_陈守真\_txt="/src/ME/sample/fqzPage/儿科萃精--陈守真.txt";

// public static String 妇科百辩\_明\_庄履严\_txt="/src/ME/sample/fqzPage/妇科百辩-明-庄履严.txt";

// public static String 妇科秘书\_\_\_txt="/src/ME/sample/fqzPage/妇科秘书--.txt";

// public static String 妇科玉尺\_清\_沈金鳌\_txt="/src/ME/sample/fqzPage/妇科玉尺-清-沈金鳌.txt";

// public static String 妇人良方集要\_宋\_陈自明\_txt="/src/ME/sample/fqzPage/妇人良方集要-宋-陈自明.txt";

// public static String 傅青主女科\_明\_傅山\_txt="/src/ME/sample/fqzPage/傅青主女科-明-傅山.txt";

// public static String 傅青主男科重编考释\_明\_傅山\_txt="/src/ME/sample/fqzPage/傅青主男科重编考释-明-傅山.txt";

// //fyydPage

// public static String 黄帝内经\_txt="/src/ME/sample/fyydPage/黄帝内经.txt";

// public static String 景岳全书\_txt="/src/ME/sample/fyydPage/景岳全书.txt";

// public static String 房中补益\_唐\_孙思邈\_txt="/src/ME/sample/fyydPage/房中补益-唐-孙思邈.txt";

// public static String 脉经\_txt="/src/ME/sample/fyydPage/脉经.txt";

// public static String 八十一难经\_汉\_扁鹊\_txt="/src/ME/sample/fyydPage/八十一难经-汉-扁鹊.txt";

// public static String 中藏经\_txt="/src/ME/sample/fyydPage/中藏经.txt";

// public static String 金匮要略\_txt="/src/ME/sample/fyydPage/金匮要略.txt";

// //fckxPage

// public static String fckxPage8\_txt="/src/ME/sample/fckxPage/fckxPage8.txt";

// //

// public static String nankekx8\_txt="nankekx8.txt";

//// //cecil

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed1\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed1.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed2\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed2.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed3\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed3.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed4\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed4.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed5\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed5.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed6\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed6.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed7\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed7.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed8\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed8.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed9\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed9.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed10\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed10.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed11\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed11.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed12\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed12.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed13\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed13.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed14\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed14.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed15\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed15.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed16\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed16.txt";

// public static final String HarrisonsPrinciplesofInternalMedicine\_18ed17\_txt = "/src/ME/sample/cecil/HarrisonsPrinciplesofInternalMedicine-18ed17.txt";

//

////

////cache

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

public class StableHTTP{

public static final String PORT\_DATABASE= "8000";

public static final String PORT\_FRONTEND= "80";

public static final String PORT\_CACHE= "6379";

public static final String PORT\_BACKEND= "8080";

public static final String PORT\_BLUETOOTH= "8008";

public static final String PORT\_TELPORT= "8888";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

import java.util.HashMap;

import java.util.Map;

//思想 罗瑶光 DNA元基催化与肽计算

//编码 罗瑶光

//#### 元基数字 = 元基符号= 肽展公式元基数字变换 = (肽概率展开数字逻辑集合) #### 1位 ##### E = I = I =(I)

//##### F = U = I++ OR Q-- =(I, Q)

//##### G = Q = Q =(Q)

//#### 1~2位

//##### 0 = D = DD =(D, DD)

//#### 2位

//##### 1 = C = DI =(DI)

//##### 3 = E = IU, DU =(IU, DU)

//##### D = V = UQ =(UQ)

//##### 9 = S = QI =(QI)

//#### 2~4位

//##### 4 = H = (IU, DU) OR DI =(IU, DU, DI) OR (IUDI, DUDI)

//#### 4位

//##### 2 = P = (IU, DU) + DI =(IUDI, DUDI)

//##### A = O = (IU, DU) + QI =(IUQI, DUQI)

//##### 7 = A = UQQI =(UQQI)

//#### 4~6位

//##### 5 = HC- = ((IU, DU) OR DI) + DI =(IUDI, DUDI, DIDI) OR (IUDIDI, DUDIDI)

//##### B = HE+ = ((IU, DU) OR DI) + (IU, DU) =(IUIU, IUDU, DUIU, DUDU, DIIU, DIDU)

//OR (IUDIIU, IUDIDU, DUDIIU, DUDIDU)

//#### 6~8位

//##### 8 = M = ((IU, DU) OR DI) + DI + QI =(IUDIQI, DUDIQI, DIDIQI)

//OR (IUDIDIQI, DUDIDIQI)

//##### 6 = X = UQ + ((IU, DU) OR DI) + DI =(UQIUDI, UQDUDI, UQDIDI)

//OR (UQIUDIDI, UQDUDIDI)

//##### C = T = UQ + ((IU, DU) OR DI) + (IU, DU) =(UQIUIU, UQIUDU, UQDUIU, UQDUDU, UQDIIU

//, UQDIDU) OR (UQIUDIIU, UQIUDIDU, UQDUDIIU, UQDUDIDU)

public interface StableMapsInitons{

public static final Map<String, String> initonsMap= new HashMap<>();

public static final Map<String, Integer> numberSet= new HashMap<>();

public static final Map<String, String> initonsSet= new HashMap<>();

public static final Map<String, String> initonsCode= new HashMap<>();

public static void init() {

initonsMap.put("A", "7");

initonsMap.put("O", "A");

initonsMap.put("P", "2");

initonsMap.put("M", "8");

initonsMap.put("V", "D");

initonsMap.put("E", "3");

initonsMap.put("C", "1");

initonsMap.put("S", "9");

initonsMap.put("I", "E");

initonsMap.put("D", "0");

initonsMap.put("U", "F");

initonsMap.put("Q", "G");

initonsMap.put("T", "C");

initonsMap.put("X", "6");

initonsMap.put("+", "B");

initonsMap.put("-", "5");

initonsMap.put("H", "4");

numberSet.put("0", 0);

numberSet.put("1", 1);

numberSet.put("2", 2);

numberSet.put("3", 3);

numberSet.put("4", 4);

numberSet.put("5", 5);

numberSet.put("6", 6);

numberSet.put("7", 7);

numberSet.put("8", 8);

numberSet.put("9", 9);

numberSet.put("A", 10);

numberSet.put("B", 11);

numberSet.put("C", 12);

numberSet.put("D", 13);

numberSet.put("E", 14);

numberSet.put("F", 15);

numberSet.put("G", 16);

initonsSet.put("0", "0");

initonsSet.put("1", "1");

initonsSet.put("2", "2");

initonsSet.put("3", "3");

initonsSet.put("4", "4");

initonsSet.put("5", "5");

initonsSet.put("6", "6");

initonsSet.put("7", "7");

initonsSet.put("8", "8");

initonsSet.put("9", "9");

initonsSet.put("10", "A");

initonsSet.put("11", "B");

initonsSet.put("12", "C");

initonsSet.put("13", "D");

initonsSet.put("14", "E");

initonsSet.put("15", "F");

initonsSet.put("16", "G");

initonsCode.put("0", "D");

initonsCode.put("1", "C");

initonsCode.put("2", "P");

initonsCode.put("3", "E");

initonsCode.put("4", "H");

initonsCode.put("5", "-");

initonsCode.put("6", "X");

initonsCode.put("7", "A");

initonsCode.put("8", "M");

initonsCode.put("9", "S");

initonsCode.put("A", "O");

initonsCode.put("B", "+");

initonsCode.put("C", "T");

initonsCode.put("D", "V");

initonsCode.put("E", "I");

initonsCode.put("F", "U");

initonsCode.put("G", "Q");

}

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

import java.util.HashMap;

import java.util.Map;

//StableToken.ETL\_SYMBOL\_SMILL

//集合下shell 计算符号

//我思考了下如果用中文设计变量，一旦出现乱码，那么中文变量对应中文字符串都是乱码，不好修复，于是用英文做变量先。

//罗瑶光

public class StableShellETL{

//稍后合并

public static final String SHELL\_ETL\_FINISH= "完成";

public static final String SHELL\_ETL\_CONFIG= "配置生成";

public static final String SHELL\_ETL\_DID\_CONFIG= "configured" ; //configured

public static final String SHELL\_ETL\_MID\_SHELL= "midShell";

public static final String SHELL\_ETL\_DOWN\_SHELL= "downShell";

public static final String SHELL\_ETL\_TIN\_SHELL\_ETL= "TinShellETL";

public static final String SHELL\_ETL\_JOIN\_BASE\_NAME= "joinBaseName";

public static final String SHELL\_ETL\_SLASH= "/";

public static final String SHELL\_ETL\_CONDITION= "condition";

public static final String SHELL\_ETL\_CULUMNNAME= "culumnName";

public static final String SHELL\_ETL\_ROWVALUE= "rowValue";

public static final String SHELL\_ETL\_SMALL= "<";

public static final String SHELL\_ETL\_LESS\_THAN= "-lt";

//作者思维 这个缩写写法来自 linux 的 c shell

public static final String SHELL\_ETL\_LESS\_AND\_EQUAL\_TO= "<=";

public static final String SHELL\_ETL\_LESS\_AND\_EQUAL\_TO\_R= "=<";

public static final String SHELL\_ETL\_LESS\_THAN\_AND\_EQUAL\_TO= "-lte";

//作者思维 这个缩写写法来自 linux 的 c shell

public static final String SHELL\_ETL\_DOUBLE\_EQUALS= "==";

public static final String SHELL\_ETL\_SIMPLE\_EUQALS= "=";

public static final String SHELL\_ETL\_TRIPAL\_EQUALS= "===";

//作者思维 这个缩写写法来自 javascript的 全等于

public static final String SHELL\_ETL\_GREATER\_AND\_EQUAL\_TO= ">=";

public static final String SHELL\_ETL\_GREATER\_AND\_EQUAL\_TO\_R= "=>";

public static final String SHELL\_ETL\_GREATER\_THAN\_AND\_EQUAL\_TO= "-gte";

//作者思维 这个缩写写法来自 linux 的 c shell

public static final String SHELL\_ETL\_GREATER= ">";

public static final String SHELL\_ETL\_GREATER\_THAN= "-gt";

//作者思维 这个缩写写法来自 linux 的 c shell

public static final String SHELL\_ETL\_STRING\_LENGTH\_GREATER\_THAN= "字符串长度大于";

public static final String SHELL\_ETL\_STRING\_LENGTH\_LESS\_THAN= "字符串长度小于";

public static final String SHELL\_ETL\_NOT\_EUQAL\_TO= "!=";

public static final String SHELL\_ETL\_NOT\_EUQAL\_TO\_R= "=!";

public static final String SHELL\_ETL\_CONTANS= "包含";

public static final String SHELL\_ETL\_FILTER\_BY= "过滤掉";

public static final String SHELL\_ETL\_DOES\_NOT\_CONTANS= "不包含";

public static final String SHELL\_ETL\_EQUAL= "equal";

public static final String SHELL\_ETL\_DOES\_NOT\_EQUALS= "!equal";

public static final String SHELL\_ETL\_IN= "in"; //作者思维 这个单词来自relational SQL 的 in

public static final String SHELL\_ETL\_COMDOT= ",";

public static final String SHELL\_ETL\_NOT\_IN= "!in";

public static Map<String, String> storeValue= new HashMap<>();

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

public interface StableString {

public static final 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。";

public static final 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。\r\n"

+ " 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\r\n"

+ "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 dp/sq(2\*pi) \* pof(m)(p) e\*\*ipt 与 dp/sq(2\*pi) \* pof (p) e\*\*ipt 正交\r\n"

+ "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\r\n"

+ " => (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"

+ "-》\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\r\n"

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

+ "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|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。。\r\n"

+ "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\r\n"

+ "|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。\r\n"

+ "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\r\n"

+ "同时，很高兴，我通过错误的伪命题得到一个真确答案：\r\n"

+ " |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.。。。。\r\n"

+ "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？\r\n"

+ "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。\r\n"

+ "而公式就是 |tero(x)>+|tcol(x)> = deta (t1-t0)\*m(t）/ deta1(t1-t0)\*p(t) \r\n"

+ "\r\n"

+ "这个公式我思考了一段时间，很明显，时间是一个带有引力波常量系数的公式。三维世界有三维世界关于时间的计算方程，高维肯定有高维的时间计算方式。\r\n"

+ "\r\n"

+ "\r\n"

+ "罗瑶光 2018年5月1日于浏阳\r\n"

+ "罗瑶光 2019年2月14日修改于浏阳\r\n"

+ "\r\n"

+ "时函数在量子物理中应用。2019-3-2\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，\r\n"

+ "最近有看了一下，\r\n"

+ "又\r\n"

+ "稍微变化了一下： \r\n"

+ "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"

+ "\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。\r\n"

+ "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。\r\n"

+ "不知道假设是不是正确的。\r\n"

+ "\r\n"

+ "我在思考这种固定的观测态怎么模拟出来。\r\n"

+ "\r\n"

+ "时函数在量子物理中应用2 2019-3-12 \r\n"

+ "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\r\n"

+ "\r\n"

+ "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。\r\n"

+ "\r\n"

+ "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。\r\n"

+ "\r\n"

+ "时函数3.指数加速度 2019-3-18 09:15\r\n"

+ " 如果瞬间转移的条件成立，那么速度怎么测量呢？\r\n"

+ "\r\n"

+ "我思考了很久，这种坍塌牵引力根据动能的大小会产生一个只有一种成指数的加速度才能解释。这种指数加速度怎么推导呢？ 我又茫目了。罗氏猜想 这个 指数加速度 和宇宙膨胀的速度通用一个常量系数。\r\n"

+ "\r\n"

+ "我有3个 想到的公式可参考。";

public static final String text3 = "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?\"\r\n"

+ "+ \"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?样，新的智能体系标准都是被需求自然择出来，这就是我要表达的中心想。\\r\\n\" + \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?些企业将产品拼命的重写升级，因为核心?发的年龄老化，新的改造无法掌握原始开发想和理论，?后产品的品质遭受巨大的冲击，\"\r\n"

+ "+ \"损失惨重，一种新的软件开发理论需要被人所证实，这也就是我的想。软件也?样，?要有自我的人工择的进化体系\\r\\n\" + \r\n"

+ "\"通过?近的 UNICORN AI\\r\\n\" + \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?，我就发现JAVA的继承没有达到具有进化想的语?标准?\"\r\n"

+ "+ \"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?要改进，比如你抽象了?个父类，而你的子类的?\"\r\n"

+ "+ \"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，么让OBJECT 得到孙类？（我的用的是OBJECT 子类继承父类，然后OBJECT \"\r\n"

+ "+ \"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，\"\r\n"

+ "+ \"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?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\" + \r\n"

+ "\"通过上面的描述，我有自己的看法，可是我还是择了JAVA, 即使繁琐，但是没有任何错误，因为用底层语?来实现就会更加繁琐陷阱更多\\r\\n\" + \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\"\r\n"

+ "+ \"为数据分析语?的一个真实的例子。C#在这个问题里?直在改进自己，类似JAVA?样，甚至和JAVA?样，可是没有?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#.?以C#的最大优势还是仅仅在WINDOWS上的控件应用。\\r\\n\" + \r\n"

+ "\"通过这段的描述，仅仅证明任何?种语?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因为他们最原始的创造理论，体系和想结构就不?样如果真?\"\r\n"

+ "+ \"JAVA和C#不，?后，通过进化的想预测，JAVA?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\" + \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?求关系的制约，导致创造想的缺乏，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\"\r\n"

+ "+ \"，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\"\r\n"

+ "+ \"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\" + \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想。然后评论一下子类变异\\r\\n\" + \r\n"

+ "\"JAVA处理子类函数是比较完美的，用过JAVA?发大型项目的人都相当有经验处理接口和继承。可? JAVA有没有变异的特呢？可以说无，比如我举个例子，当父类PUBLIC 属??1=0；，子类就无\"\r\n"

+ "+ \"法在PUBLIC 属??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\"\r\n"

+ "+ \"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? 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\"\r\n"

+ "+ \"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\" + \r\n"

+ "\"?后过上述的语?进化思想，程序进化想的表述，我有?个很深的体会。每?种语?要根深蒂固，?要有它的?求，它的功能在需求中要有选择的得到进化不然，这就是语?被淘汰的?大原因，我不喜欢?\"\r\n"

+ "+ \"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?大诟病?? 其次，语?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\"\r\n"

+ "+ \"本语?，灵活，方便。\\r\\n\" + \r\n"

+ "\"那么软件呢？软件也一样，选择?门应自己?求的语言来设计尤为重要?? 其次，软件的架构要有松度，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?\"\r\n"

+ "+ \"?致的，虽然API设计风格不一样，但是效果都很笃厚。\\r\\n\" + \r\n"

+ "\"生物?要有达尔文想，人工智能同样也存在，这是需求持久化的基?。这也是我研发UNICORN AI平台的基本条件";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

public interface StableTag {

//APP.JAVA

public static final String columnTitle\_0 = "ID";

public static final String columnTitle\_1 = "打分";

public static final String columnTitle\_2 = "中药名称";

public static final String columnTitle\_3 = "笔记原文";

public static final String columnTitle\_4 = "功效";

public static final String columnTitle\_5 = "风险规避";

public static final String columnTitle\_6 = "用法";

public static final String columnTitle\_7 = "性味";

public static final String columnTitle\_8 = "脉络";

public static final String columnTitle\_9 = "中医馆药理";

public static final String columnTitle\_10 = "经解";

public static final String columnTitle\_11 = "崇源";

public static final String columnTitle\_12 = "愚按";

public static final String columnTitle\_13 = "搭配";

public static final String columnTitle\_14 = "常见药";

public static final String NEI\_RONG = "内容";

public static final String TIAN\_JIA\_DAO\_BIAN\_JI\_YE = "添加到编辑页";

public static final String ZI\_MING = "名";

public static final String ZI\_DONG = "动";

public static final String ZI\_XING = "形";

public static final String ZI\_FU = "副";

public static final String STRING\_EMPTY = "";

public static final String STRING\_SPACE = " ";

// public Object[] columnTitle= {"ID", "打分", "中药名称", "笔记原文", "功效", "风险规避", "用法"

// , "性味", "脉络", "中医馆药理", "经解", "崇源", "愚按", "搭配", "常见药"};

// public Object[] columnTitle= {columnTitle\_0, columnTitle\_1, columnTitle\_2, columnTitle\_3, columnTitle\_4, columnTitle\_5, columnTitle\_6

// , columnTitle\_7, columnTitle\_8, columnTitle\_9, columnTitle\_10, columnTitle\_11, columnTitle\_12, columnTitle\_13, columnTitle\_14};

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

//StableToken.ETL\_SYMBOL\_SMILL

public class StableToken{

//稍后合并

//session token

public static final String EMPTY\_STRING= "";

public static final String LINE\_ENTER= "\r\n";

public static final String DNA\_TOKEN\_LOCK= "LOCK";

public static final String DNA\_TOKEN\_KEY= "KEY";

public static final String DNA\_TOKEN\_PDE= "PDE";

public static final String DNA\_TOKEN\_PDS= "PDS";

//etl node token

public static final String ETL\_NODE\_COORDINATION\_X= "NodeCoordinationX";

public static final String ETL\_NODE\_NAME= "NodeName";

public static final String ETL\_NODE\_COORDINATION\_Y= "NodeCoordinationY";

public static final String ETL\_NODE\_ID= "NodeID";

public static final String ETL\_FLASH= "flash";

public static final String ETL\_BECONNECT= "beconnect";

public static final String ETL\_LEFT\_CHOOSE= "leftChoose";

public static final String ETL\_RIGHT\_CHOOSE= "rightChoose";

public static final String ETL\_TBECONNECT= "tBeconnect";

public static final String ETL\_TBECONNECT\_X= "tBeconnectX";

public static final String ETL\_TBECONNECT\_Y= "tBeconnectY";

public static final String ETL\_TBECONNECT\_NAME= "tBeconnectName";

public static final String ETL\_TBECONNECT\_ID= "tBeconnectID";

public static final String ETL\_TBECONNECT\_PRIMARY\_KEY= "tBeconnectPrimaryKey";

public static final String ETL\_MBECONNECT= "mBeconnect";

public static final String ETL\_MBECONNECT\_X= "mBeconnectX";

public static final String ETL\_MBECONNECT\_Y= "mBeconnectY";

public static final String ETL\_MBECONNECT\_NAME= "mBeconnectName";

public static final String ETL\_MBECONNECT\_ID= "mBeconnectID";

public static final String ETL\_MBECONNECT\_PRIMARY\_KEY= "mBeconnectPrimaryKey";

public static final String ETL\_DBECONNECT= "dBeconnect";

public static final String ETL\_DBECONNECT\_X= "dBeconnectX";

public static final String ETL\_DBECONNECT\_Y= "dBeconnectY";

public static final String ETL\_DBECONNECT\_NAME= "dBeconnectName";

public static final String ETL\_DBECONNECT\_ID= "dBeconnectID";

public static final String ETL\_DBECONNECT\_PRIMARY\_KEY= "dBeconnectPrimaryKey";

public static final String ETL\_PRIMARY\_KEY= "primaryKey";

public static final String ETL\_NODE\_CONFIGURATION= "nodeConfiguration";

public static final String ETL\_ISCONFIGED= "isConfiged";

public static final String ETL\_ISEXECUTED= "isExecuted";

public static final String ETL\_SYMBOL\_PER= ":";

public static final String ETL\_SYMBOL\_NULL= "null";

public static final String ETL\_SYMBOL\_FALSE= "false";

public static final String ETL\_SYMBOL\_SMILL= ">\_<";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

public interface StableVision {

public static int[][] diaMask= new int[][] {

{1, 0, 1, 0, 1, 0, 1, 0, 1}

,{0, 1, 0, 1, 0, 1, 0, 1, 0}

,{1, 0, 1, 0, 1, 0, 1, 0, 1}

,{0, 1, 0, 1, 0, 1, 0, 1, 0}

,{1, 0, 1, 0, 1, 0, 1, 0, 1}

,{0, 1, 0, 1, 0, 1, 0, 1, 0}

,{1, 0, 1, 0, 1, 0, 1, 0, 1}

,{0, 1, 0, 1, 0, 1, 0, 1, 0}

,{1, 0, 1, 0, 1, 0, 1, 0, 1}};

//模式识别1 识别上眼睛

public static int[][] eyeHeart= new int[][] {

{ 2, 0, 0, 1, 1, 2 },

{ 0, 0, 0, 1, 1, 1 },

{ 0, 0, 0, 1, 1, 1 },

{ 0, 0, 0, 1, 1, 1 },

{ 0, 0, 0, 1, 1, 1 },

{ 2, 0, 0, 1, 1, 2 }

};

//模式识别1 识别上眼睛

public static int[][] eye= new int[][] {

{12,12,12,12,12,12,12,19,19,19,19,19,19,19,19,19,19, 9, 9, 9,11,11,11,11,11},

{12,12,12,12,12,12,12,19,19,19,19,19,19,19,19,19,19, 9, 9, 9, 9,11,11,11,11},

{12,12,12,12,12,12,12,19,19,19,19,19,19,19,19,19,19, 9, 9, 9, 9, 9, 9,11,11},

{12,12,12,12,12,12,12,19,19,19,19,19,19,19,19,19,19, 9, 9, 9, 9, 9, 9, 9,11},

{12,12,12,12,12,12,12,19,19,19,19,19,19,19,19,19,19, 9, 9, 9, 9, 9, 9, 9, 9},

{12,12,12,12,12,12,12,12,12,14,14,14,14,14,14,19,19, 9, 9, 9, 9, 9, 9, 9, 9},

{12,12,12,12,12,12,12,12,13,14,14,14,14,14,14,14,10,10, 9, 9, 9, 9, 9, 9, 9},

{12,12,12,12,12,12,12,13,13,14,14,14,14,14,14,10,10,10,10,10, 9, 9, 9, 9, 9},

{4 ,12,12,12,12,12,12,13,13,20,20, 1, 1,14,14,10,10,10,10,10,10,10, 9, 9, 9},

{2 , 2, 2, 2, 2,12,13,13,20,20,20, 1, 1, 1, 10,10,10,10,10,10,10,10,10, 9, 9},

{2 , 2, 2, 2, 2,12,13,20,20,20,20, 1, 1, 1, 1,10,10,10,10,10,10,10, 9, 9, 9},

{2 , 2, 2, 2, 2,12,13,20,20,20,20, 1, 1, 1, 1,10,10,10,10,10,10, 3, 3, 3, 3},

{2 , 2, 2, 2, 2,12,13,20,20,20,20, 1, 1, 1, 1,10,10,10,10,10, 3, 3, 3, 3, 3},

{2 , 2, 2, 2, 2,12,13,20,20,20,20, 1, 1, 1, 1,10,10,10,10,10, 3, 3, 3, 3, 3},

{2 , 2, 2, 2, 2, 2,13,13,20,20,20, 1, 1, 1,10,10,10,10,10, 3, 3, 3, 3, 3, 3},

{2 , 2, 2, 2, 2, 2,13,13,13,20,20, 1, 1,15,15,10,10,10,10, 3, 3, 3, 3, 3, 3},

{5 , 2, 2, 2, 2, 2, 2,13,13,15,15,15,15,15,15,15,10,10, 3, 3, 3, 3, 3, 3, 3},

{5 , 2, 2, 2, 2, 2, 2, 2,13,15,15,15,15,15,15,15,10, 3, 3, 3, 3, 3, 3, 3, 3},

{5 , 5, 2, 2, 2,22,22, 2, 2,15,15,15,15,15,15,15, 3, 3, 3, 3, 3, 3, 3, 3, 8},

{5 , 5, 4, 2, 2,22,22,22, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3},

{5 , 5, 5, 5, 2,22,22,22, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 7, 3, 3},

{5 , 5, 5, 5, 5,22,22,22, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 7, 3, 3},

{5 , 5, 5, 5, 5, 5,22, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 7, 7, 7, 7, 3},

{5 , 5, 5, 5, 5, 5, 5, 6, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 7, 7, 7, 7, 7, 7},

{5 , 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 3, 3, 3, 3, 3, 3, 7, 7, 7, 7, 7, 7, 7}

};

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

package SVQ.stable;

public interface StableWeb {

//DB

public static final String DB\_BASE\_NAME = "baseName";

//LOGIN

public static final String LOGIN\_TOKEN = "token";

public static final String LOGIN\_EMAIL = "email";

public static final String LOGIN\_AUTH = "auth";

//STRING

public static final String STRING\_EMPTY = "";

public static final String STRING\_SPACE\_ENTER = " \n";

public static final String STRING\_ENTER = "\n";

public static final String STRING\_QUATE = ".";

public static final String STRING\_JUNCTION = "&";

//BUFFER RANGE

public static final int BUFFER\_RANGE\_MAX = 1024;

//REST

public static final String REST\_GET\_DB\_CATEGORY = "/getDBCategory";

public static final String REST\_GET\_ALL\_DB\_CATEGORY = "/getAllDBCategory";

//SLEEPERS

public static final int SLEEPERS\_RANGE = 500;

//TCP

public static final String TCP\_PORT = "port";

public static final String STRING\_SPACE = " ";

public static final String STRING\_SLASH\_QUESTION = "\\?";

//MATH

public static final String MATH\_EQUAL = "=";

//HTTP

public static final int HTTP\_500 = 500;

public static final int HTTP\_400 = 400;

public static final int HTTP\_200 = 200;

public static final int HTTP\_404 = 404;

public static final int HTTP\_300 = 300;

//CHARSET

public static final String CHARSET\_UTF\_8 = "UTF-8";

public static final String CHARSET\_UTF8 = "UTF8";

public static final String CHARSET\_GBK = "GBK";

//FILE FORMAT

public static final String FILE\_EOT = ".eot";

public static final String FILE\_SVG = ".svg";

public static final String FILE\_OTF = ".otf";

public static final String FILE\_WOFF = ".woff";

public static final String FILE\_WOFF2 = ".woff2";

public static final String FILE\_TTF = ".ttf";

public static final String FILE\_PNG = ".png";

public static final String FILE\_JPG = ".jpg";

public static final String FILE\_JPEG = ".jpeg";

public static final String FILE\_WAV = ".wav";

public static final String FILE\_GIF = ".gif";

public static final String FILE\_JS = ".js";

public static final String FILE\_CSS = ".css";

public static final String FILE\_HTML = ".html";

public static final String FILE\_PDF = ".pdf";

public static final String FILE\_ZIP = ".zip";

public static final String FILE\_RAR = ".rar";

public static final String FILE\_XML = ".xml";

public static final String FILE\_TXT = ".txt";

//FILE Stream

public static final String STREAM\_BUFFER = "buffer";

public static final String STREAM\_BYTES = "bytes";

public static final String STREAM\_BYTES\_BUFFER = "bytesBuffer";

public static final String STREAM\_REST = "rest";

public static final String STREAM\_BYTES\_WITHOUT\_ZIP = "bytesWithoutZip";

//HTTP HEADER

public static final String HEADER\_CONTENT\_TYPE\_PDF

= "content-type: application/pdf \n\n";

public static final String HEADER\_CONTENT\_TYPE\_ZIP

= "content-type: application/zip \n\n";

public static final String HEADER\_CONTENT\_TYPE\_RAR

= "content-type: application/rar \n\n";

public static final String HEADER\_CONTENT\_TYPE\_XML

= "content-type: application/xml \n\n";

public static final String HEADER\_CONTENT\_TYPE\_TXT = "content-type: text/plain \n\n";

public static final String HEADER\_CONTENT\_TYPE\_PNG = "Content-Type: image/png \n\n";

public static final String HEADER\_CONTENT\_TYPE\_JPEG = "Content-Type: image/jpeg \n\n";

public static final String HEADER\_CONTENT\_TYPE\_JPG = "Content-Type: image/jpg \n\n";

public static final String HEADER\_CONTENT\_TYPE\_GIF = "Content-Type: image/gif \n\n";

public static final String HEADER\_CONTENT\_TYPE\_CSS = "Content-Type: text/css \n\n";

public static final String HEADER\_CONTENT\_TYPE\_HTML = "Content-Type: text/html \n\n";

public static final String HEADER\_CONTENT\_TYPE\_WAV = "Content-Type: audio/wav \n\n";

public static final String HEADER\_CONTENT\_TYPE\_FONT\_WOFF

= "Content-Type: image/font-woff \n\n";

public static final String HEADER\_CONTENT\_TYPE\_JS

= "content-type: text/javascript; charset:UTF-8 \n\n";

public static final String HEADER\_CACHE\_CONTROL = "Cache-control: max-age=315360000 \n";

public static final String HEADER\_HTTP\_200\_OK = "http/1.1 200 ok \n";

public static final String HEADER\_HTTP\_200\_OK\_DOUBLE\_ENTER ="HTTP/1.1 200 OK\n\n";

public static final String HEADER\_HOST = "Host:deta software \n";

public static final String HEADER\_CONTENT\_ENCODING\_GZIP = "Content-Encoding:gzip \n";

public static final String HEADER\_ACCEPT\_RANGES\_BYTES = "Accept-Ranges: bytes \n";

public static final String HEADER\_CONTENT\_LENGTH = "Content-Length: ";

//REST PATH

public static final String REST\_PATH\_SELECT = "/select";

public static final String REST\_PATH\_SETDB = "/setDB";

public static final String REST\_PATH\_INSERT = "/insert";

public static final String REST\_PATH\_DELETE = "/delete";

public static final String REST\_PATH\_UPDATE = "/update";

public static final String REST\_PATH\_DB\_CATEGORY = "DBCategory";

public static final String REST\_PATH\_EXEC\_DETA\_PLSQL = "/execDetaPLSQL";

public static final String REST\_PATH\_LOGIN = "/login";

public static final String REST\_PATH\_FIND = "/find";

public static final String REST\_PATH\_LOGOUT = "/logout";

public static final String REST\_PATH\_REGISTER = "/register";

public static final String REST\_PATH\_CHANGE = "/change";

public static final String REST\_PATH\_CHECK\_STATUS = "/checkStatus";

public static final String REST\_PATH\_SET\_DB\_PATH = "/setDBPath";

public static final String REST\_PATH\_SET\_DB\_TABLE = "/setDBTable";

public static final String REST\_PATH\_DELETE\_ROWS\_BY\_TABLE\_PATH\_AND\_INDEX

= "/deleteRowByTablePathAndIndex";

public static final String REST\_PATH\_INSERT\_ROW\_BY\_BASE\_NAME = "/insertRowByBaseName";

public static final String REST\_PATH\_INSERT\_ROW\_BY\_TABLE\_PATH

= "/insertRowByTablePath";

public static final String REST\_PATH\_SELECT\_ROWS\_BY\_ATTRIBUTE

= "/selectRowsByAttribute";

public static final String REST\_PATH\_SELECT\_ROWS\_BY\_TABLE\_PATH

= "/selectRowsByTablePath";

public static final String REST\_PATH\_UPDATE\_ROW\_BY\_TABLE\_PATH\_AND\_INDEX

= "/updateRowByTablePathAndIndex";

public static final String REST\_PATH\_MEDICINEZY = "/dataZY";

public static final String REST\_PATH\_MEDICINEXY = "/dataXY";

public static final String REST\_PATH\_MEDICINEZT = "/dataZT";

public static final String REST\_PATH\_MEDICINEXT = "/dataXT";

public static final String REST\_PATH\_MEDICINEYT = "/dataYT";

public static final String REST\_PATH\_SEARCH = "/search";

}

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

FMHMMNode, 隐马尔可夫类

package AVQ.ASQ.OVQ.OSQ.VSQ.obj;

import java.util.Map;

public class FMHMMNode extends HMMNode {

public Map<String, Integer> getNext() {

return next;

}

public void I\_Next(Map<String, Integer> next) {

this.next = next;

}

private Map<String, Integer> next;

}

WordFrequency, 词频类

package AVQ.ASQ.OVQ.OSQ.VSQ.obj;

public class WordFrequency {

public String getPOS() {

return POS;

}

public void I\_POS(String POS) {

this.POS = POS;

}

public String getWord() {

return word;

}

public void I\_Word(String word) {

this.word = word;

}

public int getFrequency() {

return frequency;

}

public void I\_Frequency(int frequency) {

this.frequency = frequency;

}

public WordFrequency getLeft() {

return left;

}

public void I\_Left(WordFrequency left) {

this.left = left;

}

public WordFrequency getRight() {

return right;

}

public void I\_Right(WordFrequency right) {

this.right = right;

}

private String word;

private String POS;

private int frequency;

private WordFrequency left;

private WordFrequency right;

}

FMHMMPOS, 隐马尔可夫类

package AVQ.ASQ.OVQ.OSQ.VSQ.obj;

import java.util.Map;

public class FMHMMPOS extends HMMNode {

public Map<String, Integer> getNext() {

return next;

}

public void I\_Next(Map<String, Integer> next) {

this.next = next;

}

private Map<String, Integer> next;

}

FMHMMNode, 隐马尔可夫类

package AVQ.ASQ.OVQ.OSQ.VSQ.obj;

import java.util.Map;

public class FMHMMNode extends HMMNode {

public Map<String, Integer> getNext() {

return next;

}

public void I\_Next(Map<String, Integer> next) {

this.next = next;

}

private Map<String, Integer> next;

}

Verbal, 词汇处理类

package AVQ.ASQ.OVQ.OSQ.VSQ.obj;

@SuppressWarnings("unused")

public class Verbal{

public String getChinese() {

return chinese;

}

public void I\_Chinese(String chinese) {

this.chinese = chinese;

}

public String getEnglish() {

return english;

}

public void I\_English(String english) {

this.english = english;

}

public String getPartOfSpeech() {

return partOfSpeech;

}

public void I\_PartOfSpeech(String partOfSpeech) {

this.partOfSpeech = partOfSpeech;

}

public String getExplain() {

return explain;

}

public void I\_Explain(String explain) {

this.explain = explain;

}

public Verbal getNext() {

return next;

}

public void I\_Next(Verbal next) {

this.next = next;

}

public Verbal getPrev() {

return prev;

}

public void I\_Prev(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;

}

DemoPOSforSpecial, Demo类

package ASQ.PSU.test;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import OCI.ME.analysis.C.A;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

import OEI.ME.analysis.E.BinaryForest\_AE;

import static java.lang.System.\*;

public class DemoPOSforSpecial\_UTF8 {

public static void main(String[] args) throws IOException {

A \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

Map<String, String> nlp = ((BinaryForest\_AE)\_A).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] = "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"; //韩语的切词方法请自行扩展。本人在\_A.IV\_All(); 有展示。

for (int i = 0; i < ss.length; i++) {

System.out.println("超级变态复杂病句-->" + ss[i]);

sets = \_A.parserMixedString(ss[i].replace(" ", " "));

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("分析处理真实效果-->" + 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("");

}

}

}

DemoPOS, Demo类

package ASQ.PSU.test;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import OCI.ME.analysis.C.A;

//import OCI.ME.analysis.C.A;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

import OEI.ME.analysis.E.BinaryForest\_AE;

import static java.lang.System.\*;

public class DemoPOS {

public static void main(String[] args) throws IOException {

A \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();//.init();

Map<String, String> nlp = ((BinaryForest\_AE) \_A).getPosCnToCn();

List<String> sets = new ArrayList<>();

String[] ss = new String[37];

String[] ss1 = new String[37];

ss[0] = "";

ss[1] = "海南方向逃跑";

ss[2] = "他说的确实在?????";// 本人 联想2018年的电脑是UTF8 +GBK eclipse编码，2019年买的联想 windows10操作系统 竟然没有UTF8编码，只有eclipse 支持，结果导致，GBK和UTF8 混合开发一些高级字符变成了乱码。测试demo文档我先不管了。

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?????";

ss[8] = "为人民服?????";

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?????";

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?????";

ss[16] = "学生会游?????";

ss[17] = "研究生活水平";

ss[18] = "中国有企?????";

ss[19] = "我爱美国手球";

ss[20] = "中国喜欢?????";

ss[21] = "中国热爱?????";

ss[22] = "王军虎去广州?????";

ss[23] = "王军虎头虎脑?????";

ss[24] = "将军任命了一名中?????";

ss[25] = "产量三年中将增长两??";

ss[26] = "";

ss[27] = "我来到北京清华大?????";

ss1[0] = "";

ss1[1] = "????? ????? 方向 逃跑";

ss1[2] = "????? ????? ????? 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????? ????? ????? 应聘";

ss1[8] = "????? 人民 服务";

ss1[9] = "独立 自主 ????? 平等 互利 ????? 原则";

ss1[10] = "????? 人民 办公?????";

ss1[11] = "这事 的确 ????? ????? 下来";

ss1[12] = "????? ????? ????? 把手 ????? ?????";

ss1[13] = "????? ????? ????? ????? 起来";

ss1[14] = "学生????? 宣传 ?????";

ss1[15] = "学生 ????? 主动 完成 作业";

ss1[16] = "学生????? 游戏";

ss1[17] = "研究 生活 水平";

ss1[18] = "中国 ????? 企业";

ss1[19] = "????? ????? 美国 手球";

ss1[20] = "";

ss1[21] = "";

ss1[22] = "王军????? ????? 广州 ?????";

ss1[23] = "王军 虎头虎脑 ?????";

ss1[24] = "将军 任命 ????? ?????????? 中将";

ss1[25] = "产量 ????? ????? ????? ????? 增长 两??";

ss1[26] = "";

ss1[27] = "????? 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?????";

ss[34]= "老人家中很干?????";

ss1[33] = "????? 人家 身体 不错";

ss1[34]= "老人 ????? ????? ????? 干净";

ss[35] = "版权归属做出回应";

ss[36] = "有用户发?????";

ss1[35] = "版权 归属 ????? ????? 回应";

ss1[36] = " ????? 用户 发现";

for (int i = 0; i < ss.length; i++) {

System.out.println("超级变?复杂病?????-->" + ss[i]);

sets = \_A.parserMixedString(ss[i].replace(" ", ""));//词?分?????

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("");

}

}

}

FHMMList, 隐马尔可夫类

package OCI.SVQ.MPC.fhmm.C;

//词汇翻译系统

import java.io.IOException;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

public interface FHMMList {

void index() throws IOException;

void indexMixed() throws IOException;

void indexAll() 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();

Map<String, String> getFullCnToTt();

void indexFullCnToTt() throws IOException;

}

PillowsSet, 资源类

package OEI.SVQ.MPC.fhmm.E;

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 SVQ.stable.StablePOS;

import PEU.P.time.\*;

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 + StablePOS.INT\_ONE);

}

if(index> this.index) {

if(null== largeOrder) {

largeOrder= new PillowsSet();

largeOrder.preSmallOrder= this;

}

largeOrder.arrangePillow(index, object, pillows, depth, currentDepth + StablePOS.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= StablePOS.INT\_ZERO;

int depth= pillows >> StablePOS.INT\_ONE;

int currentDepth = StablePOS.INT\_ZERO;

PillowsSet pillowsMap= new PillowsSet();

for(int i=StablePOS.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();

}

}

}

FMHMMListOneTimeImp, 隐马尔可夫类

package OEI.SVQ.MPC.fhmm.E;

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 AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMPOS;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import OCI.SVQ.MPC.fhmm.C.FHMMList;

import PEU.P.nlp.\*;

//I will build a collection class for managing this maps. at the next version.

@SuppressWarnings("unchecked")

public class FMHMMListOneTime\_E 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> fullCnToTt;

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/ StablePOS.INT\_SIX;

Map<Long, FMHMMNode>[] maps= new HashMap[StablePOS.INT\_SIX];

Iterator<Long> iterator= this.linkedHashMap.keySet().iterator();

maps[StablePOS.INT\_ZERO]= new HashMap<>();

int index= StablePOS.INT\_ZERO;

int count= StablePOS.INT\_ONE;

while(iterator.hasNext()) {

if(StablePOS.INT\_ZERO== count++ % perRatio) {

if(index< StablePOS.INT\_FIVE) {

index++;

maps[index]= new HashMap<>();

}

}

Long key= iterator.next();

maps[index].put(key, this.linkedHashMap.get(key));

}

return maps;

}

public void indexAll() throws IOException {

studyPos= new ConcurrentHashMap<>();

posCnToCn= new ConcurrentHashMap<>();

linkedHashMap= new ConcurrentHashMap<>();

listCn= new CopyOnWriteArrayList<>();

listAb= new CopyOnWriteArrayList<>();

listKo= new CopyOnWriteArrayList<>();

listJp= new CopyOnWriteArrayList<>();

listRs= new CopyOnWriteArrayList<>();

listTt= new CopyOnWriteArrayList<>();

listEn= new CopyOnWriteArrayList<>();

listPy= new CopyOnWriteArrayList<>();

InputStream inputStream= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_CN);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, StablePOS.UTF8\_STRING));

InputStream inputStreamKorea= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_KO);

BufferedReader cReaderKorea= new BufferedReader(new InputStreamReader(inputStreamKorea, StablePOS.UTF8\_STRING));

InputStream inputStreamJapan= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_JP);

BufferedReader cReaderJapan= new BufferedReader(new InputStreamReader(inputStreamJapan, StablePOS.UTF8\_STRING));

InputStream inputStreamTrandition= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_TT);

BufferedReader cReaderTrandition= new BufferedReader(new InputStreamReader(inputStreamTrandition, StablePOS.UTF8\_STRING));

InputStream inputStreamEnglish= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_EN\_TO\_EN);

BufferedReader cReaderEnglish= new BufferedReader(new InputStreamReader(inputStreamEnglish, StablePOS.UTF8\_STRING));

InputStream inputStreamPinYin= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_PY);

BufferedReader cReaderPinYin= new BufferedReader(new InputStreamReader(inputStreamPinYin, StablePOS.UTF8\_STRING));

InputStream inputStreamRs= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_RS);

BufferedReader cReaderRs= new BufferedReader(new InputStreamReader(inputStreamRs, StablePOS.UTF8\_STRING));

InputStream inputStreamAb= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_AB);

BufferedReader cReaderAb= new BufferedReader(new InputStreamReader(inputStreamAb, StablePOS.UTF8\_STRING));

String cInputString;

String cInputStringKorea;

String cInputStringJapan;

String cInputStringTrandition;

String cInputStringEnglish;

String cInputStringPinYin;

String cInputStringRs;

String cInputStringAb;

Here:

while ((cInputString = cReader.readLine()) != null) {

cInputStringKorea= cReaderKorea.readLine();

cInputStringJapan= cReaderJapan.readLine();

cInputStringTrandition= cReaderTrandition.readLine();

cInputStringEnglish= cReaderEnglish.readLine();

cInputStringPinYin= cReaderPinYin.readLine();

cInputStringRs= cReaderRs.readLine();

cInputStringAb= cReaderAb.readLine();

listCn.add(cInputString);

if(null!= cInputStringKorea) {

listKo.add(cInputStringKorea);

//加韩语例子在这。注意词库的/后面要有标识

}

if(null!= cInputStringJapan) {

listJp.add(cInputStringJapan);

//加日语例子在这。注意词库的/后面要有标识

}

if(null!= cInputStringTrandition) {

listTt.add(cInputStringTrandition);

//繁体

if(StablePOS.INT\_ONE== cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiOne.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringTrandition

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_TWO== cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiTwo.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringTrandition

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_THREE== cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiThree.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringTrandition

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_FOUR== cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiFour.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringTrandition

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(null!= cInputStringEnglish) {

listEn.add(cInputStringEnglish);

//跟上面的同理。词库翻译下即可我就不设计了。

}

if(null!= cInputStringPinYin) {

listPy.add(cInputStringPinYin);

}

if(null!= cInputStringRs) {

listRs.add(cInputStringRs);

}

if(null!= cInputStringAb) {

listAb.add(cInputStringAb);

}

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

if(StablePOS.INT\_ONE== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiOne.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_TWO== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiTwo.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_THREE== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiThree.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_FOUR== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiFour.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.fuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_FU)) {

StableMaps.fuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.fuCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.fuCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.fuCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.fuCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.dongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DONG)) {

StableMaps.dongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.dongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.dongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.dongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.dongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.liangCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIANG)) {

StableMaps.liangCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.liangCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.liangCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.liangCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.liangCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.lianCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIAN)) {

StableMaps.lianCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.lianCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.lianCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.lianCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.lianCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.baDongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_BA\_DONG)) {

StableMaps.baDongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.baDongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.baDongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.baDongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.baDongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xianDingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XIAN\_DING)) {

StableMaps.xianDingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xianDingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xianDingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xianDingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xianDingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.mingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_MING)) {

StableMaps.mingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.mingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.mingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.mingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.mingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.daiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DAI)) {

StableMaps.daiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.daiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.daiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.daiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.daiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.jieCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_JIE)) {

StableMaps.jieCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.jieCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.jieCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.jieCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.jieCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xingRongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_RONG)) {

StableMaps.xingRongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xingRongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xingRongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xingRongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xingRongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.zhuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_ZHU)) {

StableMaps.zhuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.zhuCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.zhuCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.zhuCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.zhuCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.weiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_WEI)) {

StableMaps.weiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.weiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.weiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.weiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.weiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.shengLueCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHENG\_LUE)) {

StableMaps.shengLueCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.shengLueCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.shengLueCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.shengLueCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.shengLueCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.qingTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_QING\_TAI)) {

StableMaps.qingTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.qingTaiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.qingTaiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.qingTaiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.qingTaiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xingWeiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_WEI)) {

StableMaps.xingWeiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xingWeiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xingWeiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xingWeiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xingWeiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.shiTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHI\_TAI)) {

StableMaps.shiTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.shiTaiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.shiTaiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.shiTaiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.shiTaiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.dingMingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DING\_MING)) {

StableMaps.dingMingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.dingMingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.dingMingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.dingMingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.dingMingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

studyPos.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

posCnToCn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputString);

if(null!= cInputStringKorea) {

posCnToCn.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputStringKorea);

}

if(null!= cInputStringJapan) {

if(!posCnToCn.containsKey(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])) {

posCnToCn.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringJapan

.split(StablePOS.NLP\_SYMBO\_SLASH).length> StablePOS.INT\_ONE?cInputStringJapan

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]:"δ֪");

linkedHashMap = loopLoadForest(cInputStringJapan);

}

}

if(null!= cInputStringTrandition) {

if(!posCnToCn.containsKey(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])) {

posCnToCn.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputStringTrandition);

}

}

if(null!= cInputStringEnglish) {

posCnToCn.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase(), cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringRs) {

posCnToCn.put(cInputStringRs.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase()

, cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringAb) {

posCnToCn.put(cInputStringAb.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase()

, cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

cReader.close();

cReaderKorea.close();

cReaderJapan.close();

cReaderTrandition.close();

cReaderEnglish.close();

cReaderPinYin.close();

cReaderRs.close();

cReaderAb.close();

//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 indexMixed() throws IOException {

studyPos= new ConcurrentHashMap<>();

posCnToCn= new ConcurrentHashMap<>();

linkedHashMap= new ConcurrentHashMap<>();

listCn= new CopyOnWriteArrayList<>();

listAb= new CopyOnWriteArrayList<>();

listKo= new CopyOnWriteArrayList<>();

listJp= new CopyOnWriteArrayList<>();

listRs= new CopyOnWriteArrayList<>();

listTt= new CopyOnWriteArrayList<>();

listEn= new CopyOnWriteArrayList<>();

listPy= new CopyOnWriteArrayList<>();

InputStream inputStream= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_CN);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, StablePOS.UTF8\_STRING));

InputStream inputStreamKorea= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_KO);

BufferedReader cReaderKorea= new BufferedReader(new InputStreamReader(inputStreamKorea, StablePOS.UTF8\_STRING));

InputStream inputStreamJapan= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_JP);

BufferedReader cReaderJapan= new BufferedReader(new InputStreamReader(inputStreamJapan, StablePOS.UTF8\_STRING));

InputStream inputStreamTrandition= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_TT);

BufferedReader cReaderTrandition= new BufferedReader(new InputStreamReader(inputStreamTrandition, StablePOS.UTF8\_STRING));

InputStream inputStreamEnglish= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_EN\_TO\_EN);

BufferedReader cReaderEnglish= new BufferedReader(new InputStreamReader(inputStreamEnglish, StablePOS.UTF8\_STRING));

InputStream inputStreamPinYin= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_PY);

BufferedReader cReaderPinYin= new BufferedReader(new InputStreamReader(inputStreamPinYin, StablePOS.UTF8\_STRING));

InputStream inputStreamRs= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_RS);

BufferedReader cReaderRs= new BufferedReader(new InputStreamReader(inputStreamRs, StablePOS.UTF8\_STRING));

InputStream inputStreamAb= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_AB);

BufferedReader cReaderAb= new BufferedReader(new InputStreamReader(inputStreamAb, StablePOS.UTF8\_STRING));

String cInputString;

String cInputStringKorea;

String cInputStringJapan;

String cInputStringTrandition;

String cInputStringEnglish;

String cInputStringPinYin;

String cInputStringRs;

String cInputStringAb;

Here:

while ((cInputString = cReader.readLine()) != null) {

cInputStringKorea= cReaderKorea.readLine();

cInputStringJapan= cReaderJapan.readLine();

cInputStringTrandition= cReaderTrandition.readLine();

cInputStringEnglish= cReaderEnglish.readLine();

cInputStringPinYin= cReaderPinYin.readLine();

cInputStringRs= cReaderRs.readLine();

cInputStringAb= cReaderAb.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(null!= cInputStringPinYin) {

listPy.add(cInputStringPinYin);

}

if(null!= cInputStringRs) {

listRs.add(cInputStringRs);

}

if(null!= cInputStringAb) {

listAb.add(cInputStringAb);

}

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

if(StablePOS.INT\_ONE== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiOne.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_TWO== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiTwo.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_THREE== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiThree.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(StablePOS.INT\_FOUR== cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].length()) {

StableMaps.CiFour.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.fuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_FU)) {

StableMaps.fuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.fuCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.fuCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.fuCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.fuCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.dongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DONG)) {

StableMaps.dongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.dongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.dongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.dongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.dongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.liangCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIANG)) {

StableMaps.liangCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.liangCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.liangCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.liangCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.liangCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.lianCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIAN)) {

StableMaps.lianCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.lianCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.lianCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.lianCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.lianCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.baDongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_BA\_DONG)) {

StableMaps.baDongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.baDongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.baDongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.baDongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.baDongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xianDingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XIAN\_DING)) {

StableMaps.xianDingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xianDingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xianDingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xianDingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xianDingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.mingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_MING)) {

StableMaps.mingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.mingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.mingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.mingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.mingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} if(!StableMaps.daiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DAI)) {

StableMaps.daiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.daiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.daiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.daiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.daiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} if(!StableMaps.jieCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_JIE)) {

StableMaps.jieCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.jieCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.jieCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.jieCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.jieCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xingRongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_RONG)) {

StableMaps.xingRongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xingRongCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xingRongCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xingRongCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xingRongCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.zhuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_ZHU)) {

StableMaps.zhuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.zhuCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.zhuCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.zhuCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.zhuCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.weiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_WEI)) {

StableMaps.weiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.weiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.weiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.weiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.weiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} if(!StableMaps.shengLueCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHENG\_LUE)) {

StableMaps.shengLueCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.shengLueCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.shengLueCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.shengLueCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.shengLueCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.qingTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_QING\_TAI)) {

StableMaps.qingTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.qingTaiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.qingTaiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.qingTaiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.qingTaiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

if(!StableMaps.xingWeiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_WEI)) {

StableMaps.xingWeiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.xingWeiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.xingWeiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.xingWeiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.xingWeiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} if(!StableMaps.shiTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHI\_TAI)) {

StableMaps.shiTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.shiTaiCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.shiTaiCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.shiTaiCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.shiTaiCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} if(!StableMaps.dingMingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DING\_MING)) {

StableMaps.dingMingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

if(null!= cInputStringKorea) {

StableMaps.dingMingCi.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringJapan) {

StableMaps.dingMingCi.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringTrandition) {

StableMaps.dingMingCi.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringEnglish) {

StableMaps.dingMingCi.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

} studyPos.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

posCnToCn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputString);

if(null!= cInputStringKorea) {

posCnToCn.put(cInputStringKorea.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputStringKorea);

}

if(null!= cInputStringJapan) { if(!posCnToCn.containsKey(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])) {

posCnToCn.put(cInputStringJapan.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO],cInputStringJapan

.split(StablePOS.NLP\_SYMBO\_SLASH).length> StablePOS.INT\_ONE?cInputStringJapan

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]:"δ֪");

linkedHashMap = loopLoadForest(cInputStringJapan);

}

}

if(null!= cInputStringTrandition) {

if(!posCnToCn.containsKey(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])) {

posCnToCn.put(cInputStringTrandition.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputStringTrandition);

}

}

if(null!= cInputStringEnglish) {

posCnToCn.put(cInputStringEnglish.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase(), cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringRs) {

posCnToCn.put(cInputStringRs.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase()

, cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(null!= cInputStringAb) {

posCnToCn.put(cInputStringAb.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase()

, cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

}

cReader.close();

cReaderKorea.close();

cReaderJapan.close();

cReaderTrandition.close();

cReaderEnglish.close();

cReaderPinYin.close();

cReaderRs.close();

cReaderAb.close();

//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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_CN);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while ((cInputString = cReader.readLine()) != null) {

listCn.add(cInputString);

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

if(!StableMaps.fuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_FU)) {

StableMaps.fuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.dongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DONG)) {

StableMaps.dongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.liangCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIANG)) {

StableMaps.liangCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.lianCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_LIAN)) {

StableMaps.lianCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.baDongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_BA\_DONG)) {

StableMaps.baDongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.xianDingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XIAN\_DING)) {

StableMaps.xianDingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.mingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_MING)) {

StableMaps.mingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.daiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DAI)) {

StableMaps.daiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.jieCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_JIE)) {

StableMaps.jieCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.xingRongCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_RONG)) {

StableMaps.xingRongCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.zhuCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_ZHU)) {

StableMaps.zhuCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.weiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_WEI)) {

StableMaps.weiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.shengLueCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHENG\_LUE)) {

StableMaps.shengLueCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.qingTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_QING\_TAI)) {

StableMaps.qingTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.xingWeiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_XING\_WEI)) {

StableMaps.xingWeiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.shiTaiCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_SHI\_TAI)) {

StableMaps.shiTaiCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

if(!StableMaps.dingMingCi.containsKey(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO])&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].contains(StablePOS.NLP\_CI\_DING\_MING)) {

StableMaps.dingMingCi.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

posCnToCn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase()

listcc.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listee.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

, listfn.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

, listko.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

istjp.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listgm.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

, listsp.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listrs.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listab.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listpy.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase());

}

}

public void indexFullCnToTt() throws IOException {

Iterator<String> listcc= listCn.iterator();

Iterator<String> listtt= listTt.iterator();

fullCnToTt = new HashMap<>();

while(listcc.hasNext()&&listtt.hasNext()) {

fullCnToTt.put(listcc.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO]

listtt.next().split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_EN\_TO\_CN);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while (null!= (cInputString= cReader.readLine())) {

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

posEnToCn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase(), cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

cReader.close();

}

public void indexFn() throws IOException {

listFn= new CopyOnWriteArrayList<>();

InputStream in= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_FN);

BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_KO);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_JP);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_GM);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_SP);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_AB);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_RS);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_PY);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

while (null!= (cInputString = cReader.readLine())) {

listPy.add(cInputString);

}

cReader.close();

}

public void indexTt() throws IOException {

listTt = new CopyOnWriteArrayList<>();

InputStream in = getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_TT);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

while (null!= (cInputString = cReader.readLine())) {

listTt.add(cInputString);

}

cReader.close();

}

public void indexPosEnToEn() throws IOException {

posEnToEn = new HashMap<>();

listEn = new CopyOnWriteArrayList<>();

InputStream in = getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_EN\_TO\_EN);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while (null!= (cInputString = cReader.readLine())) {

listEn.add(cInputString);

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

posEnToEn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase(), cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].toLowerCase());

}

cReader.close();

}

public void indexEnToCn() throws IOException {

enToCn = new HashMap<>();

InputStream in = getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_EN\_TO\_CN);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while (null!= (cInputString = cReader.readLine())) {

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

enToCn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO].toLowerCase(), cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

}

cReader.close();

}

public void indexCnToEn() throws IOException {

cnToEn = new HashMap<>();

InputStream in = getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_CN\_TO\_EN);

BufferedReader cReader = new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while (null!= (cInputString = cReader.readLine())) {

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

cnToEn.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE].toLowerCase());

}

cReader.close();

}

public Map<Long, FMHMMNode> loopLoadForest(String cInputString) {

Here:

for (int i = StablePOS.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.I\_Vb(StablePOS.EMPTY\_STRING + cInputString.charAt(i));

if (i + StablePOS.INT\_ONE < cInputString.length()) {

Map<String, Integer> next = new HashMap<>();

next.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE), StablePOS.INT\_ONE);

fHHMMNode.I\_Next(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 + StablePOS.INT\_ONE < cInputString.length()) {

linkedHashMap = doCheckAndRunNeroPostFix(fFHMMNode, cInputString, i);

}

return linkedHashMap;

}

HashMap<String, Integer> HashMap = new HashMap<>();

if (i + StablePOS.INT\_ONE < cInputString.length()) {

HashMap.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE)

,StablePOS.INT\_ONE);

}

fFHMMNode.I\_Next(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(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE))) {

Map<String, Integer> map = fFHMMNode.getNext();

map.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE), StablePOS.INT\_ONE);

fFHMMNode.I\_Next(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(StablePOS.NLP\_SPASE\_REP, StablePOS.SPACE\_STRING);

StringBuilder sb= new StringBuilder();

for(int i= 0; i< string.length(); i++) {

if(StopSymbol\_UTF8.symbol.contains(StablePOS.EMPTY\_STRING+ string.charAt(i))) {

list.add(sb.toString().toLowerCase());

sb.delete(StablePOS.INT\_ZERO, sb.length());

list.add(String.valueOf(string.charAt(i)));

}else {

sb.append(string.charAt(i));

}

}

if(StablePOS.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() {

this.fullCnToPy.put("鼋", "yuan");

this.fullCnToPy.put("鹑", "chun");

this.fullCnToPy.put("鸲鹆", "qu yu");

this.fullCnToPy.put("鲚鱼", "ji yu");

this.fullCnToPy.put("鲎", "hou");

this.fullCnToPy.put("醍醐", "ti hu");

this.fullCnToPy.put("豹", "bao");

this.fullCnToPy.put("蟅", "zhe");

this.fullCnToPy.put("蛴螬", "qi cao");

this.fullCnToPy.put("蘹", "huai");

this.fullCnToPy.put("堇", "jin");

this.fullCnToPy.put("檗", "bo");

this.fullCnToPy.put("濮", "pu");

this.fullCnToPy.put("稆", "lv");

this.fullCnToPy.put("穬", "kuang");

this.fullCnToPy.put("箘", "jun");

this.fullCnToPy.put("芰", "ji");

this.fullCnToPy.put("茨菰", "ci gu");

this.fullCnToPy.put("荏", "ren");

this.fullCnToPy.put("荛", "rao");

this.fullCnToPy.put("菉", "lu");

this.fullCnToPy.put("菘", "song");

this.fullCnToPy.put("蕺", "ji");

this.fullCnToPy.put("蘩蒌", "fan cai");

return this.fullCnToPy;

}

public Map<String, String> getFullCnToTt() {

return this.fullCnToTt;

}

public void indexFullNegative() throws IOException {

fullNegative= new HashMap<>();

InputStream in= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_NEGATIVE);

BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

while (null!= (cInputString= cReader.readLine())) {

if(!fullNegative.containsKey(cInputString)) {

fullNegative.put(cInputString, StablePOS.EMPTY\_STRING);

}

}

cReader.close();

}

public void indexFullPositive() throws IOException {

fullPositive= new HashMap<>();

InputStream in= getClass().getResourceAsStream(StablePOS.WORDS\_SOURSE\_LINK\_POS\_POSITIVE);

BufferedReader cReader= new BufferedReader(new InputStreamReader(in, StablePOS.UTF8\_STRING));

String cInputString;

while (null!= (cInputString= cReader.readLine())) {

if(!fullPositive.containsKey(cInputString)) {

fullPositive.put(cInputString, StablePOS.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(StablePOS.INT\_ZERO)))){

treeLeafs= output.get(Long.valueOf(treeName.charAt(StablePOS.INT\_ZERO)));

treeLeafs.put(treeName, posCnToCn.get(treeName));

output.put(Long.valueOf(treeName.charAt(StablePOS.INT\_ZERO)), treeLeafs);

}else {

treeLeafs= new HashMap<>();

}

treeLeafs.put(treeName, posCnToCn.get(treeName));

output.put(Long.valueOf(treeName.charAt(StablePOS.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;

}

}

FMHMMListImp, 隐马尔可夫类

package OEI.SVQ.MPC.fhmm.E;

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 AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import SVQ.stable.StablePOS;

import OCI.SVQ.MPC.fhmm.C.FMHMMList;

import OEI.ME.euclid.E.Euclid\_CE;

public class FMHMMList\_E implements FMHMMList {

private Map<String, String> words;

private Map<Long, FMHMMNode> linkedHashMap;

@SuppressWarnings(StablePOS.RAW\_TYPES)

private Map<Integer, Map> linkedHashMapRoot;

@SuppressWarnings(StablePOS.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(StablePOS.WORDS\_SOURSE\_LINK\_POS\_CN\_TO\_CN);

BufferedReader cReader= new BufferedReader(new InputStreamReader(inputStream, StablePOS.UTF8\_STRING));

String cInputString;

Here:

while ((cInputString = cReader.readLine()) != null) {

if(!(!cInputString.replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING).equals(StablePOS.EMPTY\_STRING)&& cInputString.split(StablePOS.NLP\_SYMBO\_SLASH).length > StablePOS.INT\_ONE )) {

continue Here;

}

words.put(cInputString.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ZERO], cInputString

.split(StablePOS.NLP\_SYMBO\_SLASH)[StablePOS.INT\_ONE]);

linkedHashMap = loopLoadForest(cInputString);

}

cReader.close();

linkedHashMapRoot = new Euclid\_CE().mCogsEuclid(linkedHashMap);

}

public Map<Long, FMHMMNode> loopLoadForest(String cInputString) {

Here:

for (int i = StablePOS.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.I\_Vb(StablePOS.EMPTY\_STRING + cInputString.charAt(i));

if (i + StablePOS.INT\_ONE < cInputString.length()) {

Map<String, Integer> next = new ConcurrentHashMap<>();

next.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE), StablePOS.INT\_ONE);

fHHMMNode.I\_Next(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 + StablePOS.INT\_ONE < cInputString.length()) {

linkedHashMap = doCheckAndRunNeroPostFix(fFHMMNode, cInputString, i);

}

} else {

ConcurrentHashMap<String, Integer> concurrentHashMap = new ConcurrentHashMap<>();

if (i + StablePOS.INT\_ONE < cInputString.length()) {

concurrentHashMap.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE),

StablePOS.INT\_ONE);

}

fFHMMNode.I\_Next(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(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE))) {

Map<String, Integer> map = fFHMMNode.getNext();

map.put(StablePOS.EMPTY\_STRING + cInputString.charAt(i + StablePOS.INT\_ONE), StablePOS.INT\_ONE);

fFHMMNode.I\_Next(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 {

}

@Override

public void studyNewPos(String string, String posStudy) {

}

@Override

public Map<String, String> getStudyPos() {

return null;

}

@Override

public Map<String, String> getFullCnToTt() {

return null;

}

@Override

public void indexFullCnToTt() throws IOException {

}

@Override

public void indexAll() throws IOException {

}

}

POSControllerImp, 语义处理类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import OCI.ME.pos.C.POS\_C;

public class POS\_CE implements POS\_C{

public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] prefixWord){

if (!wordsForest.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

return countInputStringLength;

}

if (StableMaps.daiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())) {

countInputStringLength = parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

return countInputStringLength- StablePOS.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[StablePOS.INT\_ZERO].toString())){

if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.xingRongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

||StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

addFixWordsOfTwo(charPosition, inputString, fixWord);

if (StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.fuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ZERO))){

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfTwo(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

return countInputStringLength;

}

public void addFixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {

fixWords[StablePOS.INT\_ONE].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ONE].length());

if (charPosition+ StablePOS.INT\_SEVEN < inputString.length()) {

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, charPosition + StablePOS.INT\_SEVEN));

return;

}

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_TWO, inputString.length()));

}

public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings

, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ZERO]);

String postNext=StablePOS.EMPTY\_STRING + strings[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ONE);

outputList.add(postNext);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.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[StablePOS.INT\_TWO])){

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.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() == StablePOS.INT\_ZERO){

didNotFindFirstChar(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) && (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.fuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) || StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) && (StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

for (int BackPosition = StablePOS.INT\_ZERO; BackPosition < fixWord[StablePOS.INT\_ONE].length(); BackPosition++){

int[] nestCountInputStringLength = new int[StablePOS.INT\_ONE];

int result = loopCheckBackFix(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings

, nestCountInputStringLength);

if (result == StablePOS.INT\_RIGHT){

return nestCountInputStringLength[StablePOS.INT\_ZERO];

}

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength- StablePOS.INT\_ONE;

}

return countInputStringLength- StablePOS.INT\_THREE;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

return countInputStringLength- StablePOS.INT\_ONE;

}

return countInputStringLength- StablePOS.INT\_THREE;

}

public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest, int countInputStringLength, List<String> outputList

, String[] strings, int[] nestCountInputStringLength){

String charPositionAtFixWord = StablePOS.EMPTY\_STRING + fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord) && (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| StableMaps.shengLueCi.containsKey(charPositionAtFixWord)|| StableMaps.fuCi.containsKey(charPositionAtFixWord))){

if(!wordsForest.get(fixWord[StablePOS.INT\_ZERO].toString()).contains(StablePOS.NLP\_CI\_SHENG\_LUE)

&& wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_FU)){

return StablePOS.INT\_ERROR;

}

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord

, Map<String, String> wordsForest){

if(!wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

if(wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

}

}

public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings, StringBuilder[] fixWord){

outputList.add(strings[StablePOS.INT\_ONE]);

outputList.add(strings[StablePOS.INT\_THREE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_THREE]);

return countInputStringLength;

}

public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()){

didNotFindFirstChar(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(wordsForest.containsKey(strings[StablePOS.INT\_TWO])) {

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

} else if(fixWord[StablePOS.INT\_ONE].length()> StablePOS.INT\_ONE) {

String firstChar= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ZERO);

String secondChar= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].toString().charAt(StablePOS.INT\_ONE);

if(!StableMaps.fuCi.containsKey(firstChar)&& !StableMaps.fuCi.containsKey(secondChar)

&&!StableMaps.fuCi.containsKey(firstChar+ secondChar)) {

if(wordsForest.containsKey(firstChar)&& wordsForest.containsKey(secondChar)) {

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength= StablePOS.INT\_ONE;

if(wordsForest.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

countInputStringLength= StablePOS.INT\_THREE;

}

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return countInputStringLength- StablePOS.INT\_ONE;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

}

return countInputStringLength;

}

public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()&& (wordsForest.get(strings[StablePOS.INT\_TWO])

.contains(StablePOS.NLP\_CI\_WEI))){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

if (outputList.size() > StablePOS.INT\_ZERO&& wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.lianCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

} else{

if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())) {

if(StableMaps.xingWeiCi.containsKey(strings[StablePOS.INT\_ONE])

|| StableMaps.xingRongCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

if(StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength = StablePOS.INT\_TWO;

return countInputStringLength;

}else if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

}

}

if(StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength = parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return StablePOS.INT\_TWO;

}

if ((StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())||StableMaps.xingRongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))

&& StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

return StablePOS.INT\_TWO;

}

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

outputList.add(strings[StablePOS.INT\_ZERO]);

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return StablePOS.INT\_THREE;

}

return StablePOS.INT\_ONE;

}

public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.liangCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength = parserFirstCharOfTwoForMap(countInputStringLength, outputList, strings, fixWord

, wordsForest);

return countInputStringLength;

}

countInputStringLength -= StablePOS.INT\_TWO;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.INT\_TWO;

}

return countInputStringLength;

}

return countInputStringLength;

}

public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings

, StringBuilder[] fixWord, Map<String, String> wordsForest){

countInputStringLength -= StablePOS.INT\_TWO;

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.setFrequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength += StablePOS.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[StablePOS.INT\_ZERO].toString())){

if (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord

, wordsForest);

return countInputStringLength;

}

countInputStringLength -= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.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()== StablePOS.INT\_ZERO&& StableMaps.weiCi.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

if (outputList.size()> StablePOS.INT\_ZERO&& wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.lianCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.qingTaiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

} else{

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength+= StablePOS.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()== StablePOS.INT\_ZERO){

didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString()) && (StableMaps.mingCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.daiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.weiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.fuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

countInputStringLength = parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.zhuCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.shengLueCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString()))){

for (int BackPosition= StablePOS.INT\_ZERO; BackPosition< fixWord[StablePOS.INT\_ONE].length(); BackPosition++){

int[] nestCountInputStringLength = new int[StablePOS.INT\_ONE];

int result= loopCheckBackFixForMap(fixWord, BackPosition, wordsForest, countInputStringLength, outputList, strings

, nestCountInputStringLength);

if (result== StablePOS.INT\_RIGHT){

return nestCountInputStringLength[StablePOS.INT\_ZERO];

}

}

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength += StablePOS.INT\_TWO;

}

return countInputStringLength;

}

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

countInputStringLength+= StablePOS.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= StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ONE].charAt(backPosition);

if (wordsForest.containsKey(charPositionAtFixWord)&& (StableMaps.zhuCi.containsKey(charPositionAtFixWord)

|| wordsForest.get(charPositionAtFixWord).contains(StablePOS.NLP\_CI\_SHENG\_LUE))){

nestCountInputStringLength[StablePOS.INT\_ZERO]= parserFirstCharOfThreeForMap(countInputStringLength, outputList

, strings, fixWord, wordsForest);

return StablePOS.INT\_RIGHT;

}

return StablePOS.INT\_ERROR;

}

public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList, int countInputStringLength

, String[] strings, StringBuilder[] fixWord){

if (StablePOS.INT\_ZERO== outputList.size()){

didNotFindFirstCharForMap(outputList, strings, fixWord, wordsForest);

return countInputStringLength;

}

if (wordsForest.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThreeForMap(countInputStringLength, outputList, strings, fixWord, wordsForest);

return countInputStringLength;

} else{

countInputStringLength-= StablePOS.INT\_THREE;

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

if (outputList.containsKey(strings[StablePOS.INT\_ONE])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ONE]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ONE]);

outputList.put(strings[StablePOS.INT\_ONE], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength+= StablePOS.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[StablePOS.INT\_TWO])){

return;

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

}

}

public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings

, StringBuilder[] fixWord, Map<String, String> wordsForest){

countInputStringLength-= StablePOS.INT\_THREE;

if (outputList.containsKey(strings[StablePOS.INT\_ZERO])){

WordFrequency wordFrequency = outputList.get(strings[StablePOS.INT\_ZERO]);

wordFrequency.setFrequency(wordFrequency.getFrequency()+ StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_ZERO]);

outputList.put(strings[StablePOS.INT\_ZERO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength ++;

if (wordsForest.containsKey(strings[StablePOS.INT\_TWO])){

if (outputList.containsKey(strings[StablePOS.INT\_TWO])){

WordFrequency wordFrequency= outputList.get(strings[StablePOS.INT\_TWO]);

wordFrequency.setFrequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

} else{

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.setFrequency(StablePOS.INT\_ONE);

wordFrequency.setWord(strings[StablePOS.INT\_TWO]);

outputList.put(strings[StablePOS.INT\_TWO], wordFrequency);

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

countInputStringLength+= StablePOS.INT\_TWO;

return countInputStringLength;

}

return countInputStringLength;

}

public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord){

if (StableMaps.xingWeiCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())|| StableMaps.mingCi

.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO])){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_THREE])){

if(StableMaps.fuCi.containsKey(StablePOS.EMPTY\_STRING+ (0== fixWord[StablePOS.INT\_ONE].length()? "@^\_^@": fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO)))){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ONE])|| StableMaps.qingTaiCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

if(!StableMaps.dingMingCi.containsKey(strings[StablePOS.INT\_ZERO])){

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])|| StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])){

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

countInputStringLength= StablePOS.INT\_ONE;

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_THREE])){

if(StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])) {

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

if(null!= fixWord[StablePOS.INT\_ZERO]&& StablePOS.INT\_ZERO<fixWord[StablePOS.INT\_ZERO].length()){

if(StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING+ fixWord[StablePOS.INT\_ZERO].charAt(StablePOS.INT\_ZERO))){

if(!StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])){

if(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

if(!StableMaps.dongCi.containsKey(fixWord[StablePOS.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(StablePOS.INT\_ZERO< fixWord[StablePOS.INT\_ONE].length()&& StableMaps.dingMingCi.containsKey(StablePOS.EMPTY\_STRING

+ fixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if(StableMaps.dongCi.containsKey(fixWord[StablePOS.INT\_ZERO].toString())){

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

countInputStringLength= parserFirstTwoCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

}

countInputStringLength= parserFirstCharOfThree(countInputStringLength, outputList, strings, fixWord);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputStringLength= StablePOS.INT\_TWO;

return countInputStringLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputStringLength;

}

fixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWord[StablePOS.INT\_ZERO].length());

fixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputStringLength= StablePOS.INT\_ONE;

}

public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

if ((StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO].toString()))

&& (StableMaps.jieCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

|| StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString()))){

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO].toString())

||StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if (StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])){

outputList.add(strings[StablePOS.INT\_ONE]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ONE]);

countInputLength= StablePOS.INT\_TWO;

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_TWO])) {

outputList.add(strings[StablePOS.INT\_TWO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_TWO]);

return countInputLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return countInputLength= StablePOS.INT\_ONE;

}

public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

if (StableMaps.fuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_TWO].toString())) {

if (StableMaps.zhuCi.containsKey(StablePOS.EMPTY\_STRING+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))

|| StableMaps.daiCi.containsKey(StablePOS.EMPTY\_STRING+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))) {

countInputLength= parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

countInputLength= parserFirstTwoCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.INT\_ONE;

}

}

POSControllerCognitionImp, 语义处理类

package OEI.ME.pos.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.ME.pos.C.POS\_C;

public class POS\_C\_Cognition\_E implements POS\_C{

@Override

public int chuLiBaDongCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] prefixWord) {

return 0;

}

@Override

public int chuLiMingCiOfTwo(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord, int charPosition, String inputString) {

return 0;

}

@Override

public void I\_FixWordsOfTwo(int charPosition, String inputString, StringBuilder[] fixWords) {

}

@Override

public int parserFirstCharOfTwo(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiLianCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int loopCheckBackFix(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,

int countInputStringLength, List<String> outputList, String[] strings, int[] nestCountInputStringLength) {

return 0;

}

@Override

public void didNotFindFirstChar(List<String> outputList, String[] strings, StringBuilder[] fixWord,

Map<String, String> wordsForest) {

}

@Override

public int parserFirstCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

return 0;

}

@Override

public int parserFirstTwoCharOfThree(int countInputStringLength, List<String> outputList, String[] strings,

StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiZhuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiJieCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiLiangCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiMingCiOfTwoForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int parserFirstCharOfTwoForMap(int countInputStringLength, Map<String, WordFrequency> outputList,

String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {

return 0;

}

@Override

public int chuLiLiangCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiJieCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiLianCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int loopCheckBackFixForMap(StringBuilder[] fixWord, int backPosition, Map<String, String> wordsForest,

int countInputStringLength, Map<String, WordFrequency> outputList, String[] strings,

int[] nestCountInputStringLength) {

return 0;

}

@Override

public int chuLiZhuCiOfThreeForMap(Map<String, String> wordsForest, Map<String, WordFrequency> outputList,

int countInputStringLength, String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public void didNotFindFirstCharForMap(Map<String, WordFrequency> outputList, String[] strings,

StringBuilder[] fixWord, Map<String, String> wordsForest) {

}

@Override

public int parserFirstCharOfThreeForMap(int countInputStringLength, Map<String, WordFrequency> outputList,

String[] strings, StringBuilder[] fixWord, Map<String, String> wordsForest) {

return 0;

}

@Override

public int chuLiMingCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputStringLength,

String[] strings, StringBuilder[] fixWord) {

return 0;

}

@Override

public int chuLiShiTaiCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

return 0;

}

@Override

public int chuLiFuCiOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

return 0;

}

@Override

public int chuLiLianCiPostFixOfThree(Map<String, String> wordsForest, List<String> outputList, int countInputLength,

String[] strings, StringBuilder[] prefixWord) {

return 0;

}

}

NLPController, 自然语言处理类

package OCI.ME.nlp.C;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

public interface NLP\_C {

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doPOSAndEMMCheckOfThree(int countInputLength, List<String> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangCheck(int countInputStringLength, List<String> output, StringBuilder stringBuilder,Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangCheckForMap(int countInputStringLength, List<String> output, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils, int charPosition, String textInputString);

public int doSlangPartAndPOSCheckForTwoCharForMap(int countInputStringLength, Map<String, WordFrequency> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, Pos\_C\_XCDX\_P posUtils);

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils);

public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, Pos\_C\_XCDX\_P posUtils);

}

NLPControllerImp, 自然语言处理类

package OEI.ME.nlp.E;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import SVQ.stable.StableMaps;

import ME.utils.WordForestUtil;

import OCI.ME.nlp.C.NLP\_C;

import OCI.ME.pos.C.POS\_C;

public class NLP\_CE implements NLP\_C{

public int doSlangPartAndPOSCheckForTwoChar(int countInputStringLength, List<String> outputList

, StringBuilder stringBuilder, Map<String, String> wordsForest, StringBuilder[] prefixWord

, POS\_C posUtils, int charPosition, String textInputString){

String countWordNode= stringBuilder.toString();

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

if(StableMaps.CiTwo.containsKey(countWordNode)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

outputList.add(countWordNode);

return countInputStringLength;

}

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(stringBuilder.charAt(StablePOS.INT\_ZERO));

outputList.add(StablePOS.EMPTY\_STRING+ stringBuilder.charAt(StablePOS.INT\_ZERO));

return countInputStringLength- StablePOS.INT\_ONE;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= countWordNode;

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength= posUtils.chuLiMingCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.baDongCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputStringLength = posUtils.chuLiBaDongCiOfTwo(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO].toString())){

if (StableMaps.dongCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())){

if (!StableMaps.jieCi.containsKey(countWordNode)){

countInputStringLength= posUtils.parserFirstCharOfTwo(countInputStringLength, outputList, strings, prefixWord);

return countInputStringLength;

}

}

}

if (StableMaps.CiTwo.containsKey(countWordNode)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.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

, POS\_C posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (StableMaps.CiThree.containsKey(inputString)){

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

outputList.add(inputString);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if (!StableMaps.CiOne.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

if(StableMaps.lianCi.containsKey(strings[StablePOS.INT\_THREE])) {

countInputLength = posUtils.chuLiLianCiPostFixOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLianCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.jieCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiJieCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiZhuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength = posUtils.chuLiLiangCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.mingCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiMingCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.shiTaiCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiShiTaiCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if (StableMaps.dongCi.containsKey(strings[StablePOS.INT\_ZERO])||StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

if(StableMaps.zhuCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&& (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_TWO])

|| StableMaps.jieCi.containsKey(strings[StablePOS.INT\_TWO]))) {

countInputLength = posUtils.parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

if(StableMaps.fuCi.containsKey(strings[StablePOS.INT\_TWO])||StableMaps.mingCi.containsKey(strings[StablePOS.INT\_TWO])

||StableMaps.daiCi.containsKey(strings[StablePOS.INT\_TWO])) {

countInputLength = posUtils.parserFirstCharOfThree(countInputLength, outputList, strings, prefixWord);

return countInputLength;

}

}

if (StableMaps.fuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiFuCiOfThree(wordsForest, outputList, countInputLength, strings, prefixWord);

return countInputLength;

}

if(StableMaps.CiTwo.containsKey(strings[StablePOS.INT\_ONE])) {

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoChar(--countInputLength, outputList, stringsBuilder.append(strings[StablePOS.INT\_ONE])

, wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputLength;

}

outputList.add(strings[StablePOS.INT\_ZERO]);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(strings[StablePOS.INT\_ZERO]);

return StablePOS.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, POS\_C posUtils, int charPosition, String textInputString){

String inputString = stringBuilder.toString();

if (StableMaps.CiFour.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}//will make pre 3 or post 3 check. now finished pre 3 .20190330

String preRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE);

String inRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ONE)+ inputString.charAt(StablePOS.INT\_TWO);

String postRegister= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE);

if(StableMaps.dongCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_THREE)

+ prefixWord[StablePOS.INT\_ONE].charAt(StablePOS.INT\_ZERO))) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if (StableMaps.CiTwo.containsKey(preRegister)){

if (StableMaps.CiTwo.containsKey(postRegister)){

String string= StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO);

if(StableMaps.xingWeiCi.containsKey(prefixWord[StablePOS.INT\_ZERO].toString())

&&StableMaps.shiTaiCi.containsKey(string)) {

output.add(string);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(string);

return countInputStringLength- StablePOS.INT\_THREE;

}

if(StableMaps.zhuCi.containsKey(string)){

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE))

+ inputString.charAt(StablePOS.INT\_TWO);

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

countInputStringLength= posUtils.chuLiZhuCiOfThree(wordsForest, output, countInputStringLength-StablePOS.INT\_ONE, strings, prefixWord);

return countInputStringLength;

}

output.add(preRegister);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister);

return countInputStringLength-StablePOS.INT\_TWO;

}

}

if(StableMaps.CiThree.containsKey(preRegister+ inputString.charAt(StablePOS.INT\_TWO))&& !StableMaps.CiTwo.containsKey(postRegister)) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

output.add(preRegister+ inputString.charAt(StablePOS.INT\_TWO));

return countInputStringLength- StablePOS.INT\_ONE ;

}

if(StableMaps.CiTwo.containsKey(preRegister)&& StableMaps.CiTwo.containsKey(inRegister)) {

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

if(StableMaps.CiTwo.containsKey(preRegister)) {

countInputStringLength= doSlangPartAndPOSCheckForTwoChar(countInputStringLength- StablePOS.INT\_TWO, output

, stringBuilder.delete(StablePOS.INT\_TWO, StablePOS.INT\_FOUR), wordsForest, prefixWord, posUtils, charPosition, textInputString);

return countInputStringLength;

}

output.add(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO));

return countInputStringLength= StablePOS.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, POS\_C posUtils, int charPosition, String textInputString){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

output.add(inputString);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(inputString);

return countInputStringLength;

}

countInputStringLength= doPOSAndEMMCheckOfThree(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.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

, POS\_C posUtils){

String countWordNode= stringBuilder.toString();

if (!wordsForest.containsKey(countWordNode)){

WordForestUtil.wordsForestNotContainsKey(outputList, countWordNode, prefixWord);

return --countInputStringLength;

}

if (prefixWord[StablePOS.INT\_ZERO].length()== StablePOS.INT\_ZERO){

WordForestUtil.prefixWordEqualZero(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

String[] strings= new String[StablePOS.INT\_TWO];

strings[StablePOS.INT\_ZERO]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO))

+ String.valueOf(countWordNode.charAt(StablePOS.INT\_ONE));

if (wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

if (wordsForest.get(strings[StablePOS.INT\_ZERO]).contains(StablePOS.NLP\_CI\_MING)){

countInputStringLength= posUtils.chuLiMingCiOfTwoForMap(wordsForest, outputList, countInputStringLength

, strings, prefixWord);

return countInputStringLength;

}

}

if (wordsForest.containsKey(strings[StablePOS.INT\_ONE])){

WordForestUtil.wordsForestContainsKey(outputList, countWordNode, prefixWord);

return countInputStringLength;

}

return StablePOS.INT\_ZERO;

}

public int doPOSAndEMMCheckOfThreeForMap(int countInputLength, Map<String, WordFrequency> outputList

, Map<String, String> wordsForest, StringBuilder stringBuilder, StringBuilder[] prefixWord, POS\_C posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

String[] strings= new String[StablePOS.INT\_FOUR];

strings[StablePOS.INT\_ZERO]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO));

strings[StablePOS.INT\_ONE]= String.valueOf(inputString.charAt(StablePOS.INT\_ZERO))

+ inputString.charAt(StablePOS.INT\_ONE);

strings[StablePOS.INT\_TWO]= String.valueOf(inputString.charAt(StablePOS.INT\_ONE)

+ inputString.charAt(StablePOS.INT\_TWO));

strings[StablePOS.INT\_THREE]= String.valueOf(inputString.charAt(StablePOS.INT\_TWO));

if (null== prefixWord[StablePOS.INT\_ZERO]){

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(outputList, inputString, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (!wordsForest.containsKey(strings[StablePOS.INT\_ZERO])){

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiZhuCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.liangCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLiangCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.zhuCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiJieCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

if (StableMaps.lianCi.containsKey(strings[StablePOS.INT\_ZERO])){

countInputLength= posUtils.chuLiLianCiOfThreeForMap(wordsForest, outputList, countInputLength

, strings, prefixWord);

return countInputLength;

}

StringBuilder stringsBuilder= new StringBuilder();

countInputLength= doSlangPartAndPOSCheckForTwoCharForMap(--countInputLength, outputList

, stringsBuilder.append(strings[StablePOS.INT\_ONE]), wordsForest, prefixWord, posUtils);

return countInputLength;

}

public int doSlangCheckForMap(int countInputStringLength, Map<String, WordFrequency> output, StringBuilder stringBuilder

, Map<String, String> wordsForest, StringBuilder[] prefixWord, POS\_C posUtils){

String inputString= stringBuilder.toString();

if (wordsForest.containsKey(inputString)){

WordForestUtil.wordsForestContainsKey(output, inputString, prefixWord);

return countInputStringLength;

}

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE))) {

if(StableMaps.mingCi.containsKey(StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_TWO)+ inputString.charAt(StablePOS.INT\_THREE))) {

WordForestUtil.wordsForestContainsKey(output, StablePOS.EMPTY\_STRING+ inputString.charAt(StablePOS.INT\_ZERO)+ inputString.charAt(StablePOS.INT\_ONE), prefixWord);

return countInputStringLength;

}

}

countInputStringLength= doPOSAndEMMCheckOfThreeForMap(--countInputStringLength, output, wordsForest

, stringBuilder.delete(StablePOS.INT\_THREE, StablePOS.INT\_FOUR), prefixWord, posUtils);

return countInputStringLength;

}

}

NEROController, 神经网络索引类

package OCI.ME.nero.C;

import java.util.Map;

import SVQ.stable.StablePOS;

public interface NERO\_C {

@SuppressWarnings(StablePOS.RAW\_TYPES)

StringBuilder getBinaryForestRecurWord(StringBuilder inputStringWordNode, String inputString, int charPosition

, int inputStringLength, Map<Integer, Map> forestRoots, int forestDepth, int charPositionNext);

}

NEROControllerOneTimeImp, 神经网络索引类

package OEI.ME.nero.E;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import SVQ.stable.StablePOS;

import OCI.ME.nero.C.NERO\_C\_OneTime;

//著作权人+作者= 罗瑶光

public class NERO\_C\_OneTime\_E implements NERO\_C\_OneTime {

public StringBuilder getBinaryForestRecurWordOneTime(StringBuilder outputWordNode, String inputString

, int charPosition, int inputStringLength, Map<Long, FMHMMNode> forestRoots, int forestDepth

, int charPositionNext) {

if (StablePOS.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 (StablePOS.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 (StablePOS.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;

}

}

NEROControllerImp, 神经网络索引类

package OEI.ME.nero.E;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import SVQ.stable.StablePOS;

import OCI.ME.nero.C.NERO\_C;

public class NERO\_CE implements NERO\_C {

@SuppressWarnings({StablePOS.RAW\_TYPES, StablePOS.UNCHECKED})

public StringBuilder getBinaryForestRecurWord(StringBuilder outputWordNode, String inputString, int charPosition

, int inputStringLength, Map<Integer, Map> forestRoots, int forestDepth, int charPositionNext ) {

if (StablePOS.INT\_THREE== forestDepth) {

return outputWordNode;

}

char charAtPosition= inputString.charAt(charPosition);

int rangeHigh= charAtPosition>> StablePOS.INT\_TEN;

Map<Integer, Map> trees= forestRoots.get(rangeHigh);

if (null== trees) {

return outputWordNode;

}

int range= charAtPosition>> StablePOS.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+ StablePOS.INT\_ONE, ++charPositionNext);

}

return outputWordNode;

}

}

Quick6DLuoYaoguangSortImp, 极快速排序类

package OEI.ME.liner.E;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import OCI.ME.liner.C.Quick6DLuoYaoguangSort;

/\*

\*\* 快排6小高峰修正算法 作者 罗瑶光

\*\* 快排10小高峰修正算法 作者 罗瑶光 20200921

\*/

public class Quick6DLuoYaoguangSort\_E implements Quick6DLuoYaoguangSort {

public void quick6DLuoYaoGuangSortWordFrequency(List<WordFrequency> list, int leftPosition, int rightPosition) {

int c= rightPosition- leftPosition+ StablePOS.INT\_ONE;

if(c> StablePOS.INT\_FOUR) {

int pos = partition(list, leftPosition, rightPosition);

if(leftPosition< pos- StablePOS.INT\_ONE) {

quick6DLuoYaoGuangSortWordFrequency(list, leftPosition, pos - StablePOS.INT\_ONE);

}

if(pos+ StablePOS.INT\_ONE< leftPosition) {

quick6DLuoYaoGuangSortWordFrequency(list, ++ pos, rightPosition);

}

return;

}

int i= leftPosition;

for(int j= i+ StablePOS.INT\_ONE; j< leftPosition+ c; j= i++){

while(j> leftPosition){

if (list.get(j).getFrequency()< list.get(--j).getFrequency()) {

WordFrequency wordFrequency = list.get(j+ StablePOS.INT\_ONE);

list.set(j+ StablePOS.INT\_ONE, list.get(j));

list.set(j, wordFrequency);

}

}

}

}

public int partition(List<WordFrequency> list, int leftPosition, int 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, （下面可修改为<=号 见TopSort5 描述。）

if (wordFrequencyX.getFrequency()< wordFrequencyY.getFrequency()) {

wordFrequencyY= wordFrequencyX;

}

//小高峰修正边缘均衡结束

while (leftPositionNew++ < rightPosition) {

while (!(list.get(leftPositionNew++).getFrequency()> wordFrequencyY.getFrequency())

|| (leftPositionNew> rightPosition)){

}

while (list.get(rightPosition--).getFrequency()> wordFrequencyY.getFrequency()) {

}

if (--leftPositionNew< ++rightPosition){

WordFrequency wordFrequency= list.get(rightPosition);

list.set(rightPosition, list.get(leftPositionNew));

list.set(leftPositionNew, wordFrequency);

}

}

list.set(leftPosition, list.get(rightPosition));

list.set(rightPosition, wordFrequencyY);

return rightPosition;

}

@SuppressWarnings(StablePOS.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 StablePOS.INT\_ZERO;

}

@Override

public Map<Integer, WordFrequency> frequencyWordMapToMap(Map<String, WordFrequency> map) {

return null;

}

}

Quick6DLuoYaoguangSort3DMapImp, 极快速排序类

package OEI.ME.liner.E;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

//import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import OCI.ME.liner.C.Quick6DLuoYaoguangSort;

public class Quick6DLuoYaoguangSort3DMap\_E implements Quick6DLuoYaoguangSort {

@Override

public void quick6DLuoYaoGuangSortWordFrequency(Map<Integer, WordFrequency> map, int leftPosition,

int rightPosition) {

if (leftPosition< rightPosition) {

int c= rightPosition- leftPosition + StablePOS.INT\_ONE;

if (c< StablePOS.INT\_FOUR) {

int j;

for (int i= StablePOS.INT\_ONE+ leftPosition; i< leftPosition+ c; i++) {

j= i;

while (j>= StablePOS.INT\_ONE+ leftPosition) {

if (map.get(j).getFrequency()< map.get(j- StablePOS.INT\_ONE).getFrequency()) {

WordFrequency wordFrequency= map.get(j);

map.put(j, map.get(j- StablePOS.INT\_ONE));

map.put(j- StablePOS.INT\_ONE, wordFrequency);

}

j--;

}

}

return;

}

int pos= partition(map, leftPosition, rightPosition);

quick6DLuoYaoGuangSortWordFrequency(map, leftPosition, pos- StablePOS.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)) {

}

while (map.get(rightPosition--).getFrequency()> wordFrequencyY.getFrequency()) {

}

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(StablePOS.RAW\_TYPES)

public Map<Integer, WordFrequency> frequencyWordMapToMap(Map<String, WordFrequency> map) {

Map<Integer, WordFrequency> listMap= new HashMap<>();

Iterator iterator= map.keySet().iterator();

int c= StablePOS.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 StablePOS.INT\_ZERO;

}

public List<WordFrequency> frequencyWordMapToList(Map<String, WordFrequency> map) {

return null;

}

}

EuclidControllerImp, 欧基里德算法类

package OEI.ME.euclid.E;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import SVQ.stable.StablePOS;

import OCI.ME.euclid.C.Euclid\_C;

import java.util.HashMap;

import java.util.Iterator;

public class Euclid\_CE implements Euclid\_C {

@SuppressWarnings({StablePOS.RAW\_TYPES, StablePOS.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(StablePOS.EMPTY\_STRING + keyValue);

int range= (charOfKeyValueToInteger.intValue()>> StablePOS.INT\_SIX);

int rangeHigh= range >> StablePOS.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;

}

}

EuclidController, 欧基里德算法类

package OCI.ME.euclid.C;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

import SVQ.stable.StablePOS;

public interface Euclid\_C {

@SuppressWarnings({StablePOS.RAW\_TYPES})

Map<Integer, Map> mCogsEuclid(Map<Long, FMHMMNode> concurrentHashMap);

}

Analyzer, 分词类

package OCI.ME.analysis.C;

import java.io.IOException;

import java.util.List;

//import java.util.Map;

//

//import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

//import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

//import PEQ.AMV.ECS.test.SensingTest;

public interface A {

void IV\_() throws IOException;

void IV\_Mixed() throws IOException;

void IV\_All() throws IOException;//我给出了一个扩展多语种示例

List<String> parserString(String input);

void I\_FixWords(int charPosition, String inputString,StringBuilder[] fixWords);

String[] parserEnglishString(String englishString);

List<String> parserMixedString(String mixedString);

}

AnalyzerImp, 分词类

package OEI.ME.analysis.E;

import java.io.IOException;

import java.util.List;

import java.util.Map;

//import java.util.concurrent.ConcurrentHashMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.FMHMMNode;

//import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

//import ME.utils.WordFrequencyUtil;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A;

import OCI.ME.liner.C.Quick6DLuoYaoguangSort;

import OCI.ME.nero.C.NERO\_C\_OneTime;

//import OCI.ME.nlp.C.NLP\_C;

//import OCI.ME.pos.C.POS\_C;

//import OCI.ME.pos.C.Pos\_C\_XCDX;

import OCI.ME.pos.C.Pos\_C\_XCDX\_P;

import OCI.SVQ.MPC.fhmm.C.FHMMList;

import OEI.AVC.SUQ.SVQ.MPC.fhmm.E.EmotionMap\_E;

import OEI.ME.liner.E.Quick6DLuoYaoguangSort3DMap\_E;

import OEI.ME.nero.E.NERO\_C\_OneTime\_E;

//import OEI.ME.nlp.E.NLP\_CE;

import OEI.ME.nlp.E.Nlp\_CE\_XCDX\_S;

//import OEI.ME.pos.E.POS\_CE;

//import OEI.ME.pos.E.Pos\_CE\_XCDX;

import OEI.ME.pos.E.Pos\_CE\_XCDX\_P;

import OEI.SVQ.MPC.fhmm.E.FMHMMListOneTime\_E;

import java.util.Iterator;

import java.util.LinkedList;

import PEQ.AMV.ECS.test.SensingTest;

public class AE implements A {

protected FHMMList fHMMList;

protected NERO\_C\_OneTime neroController;

protected Nlp\_CE\_XCDX\_S nlpController;

protected Pos\_C\_XCDX\_P 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;

protected EmotionMap emotionMap;

protected SensingTest sensingTest;

public void IV\_() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.index();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

}

public void IV\_Mixed() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.indexMixed();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

fHMMList.indexFullCnToPy();

fHMMList.indexFullCnToKo();;

fHMMList.indexFullCnToJp();;

fHMMList.indexFullCnToTt();

fHMMList.indexFullCnToRs();

fHMMList.indexFullCnToAb();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

emotionMap= new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_NegativeMap();

emotionMap.IV\_PositiveMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

sensingTest= new SensingTest();

}

public void IV\_All() throws IOException {

this.fHMMList= new FMHMMListOneTime\_E();

fHMMList.indexAll();

fHMMList.indexPosEnToCn();

fHMMList.indexPosEnToEn();

fHMMList.indexEnToCn();

fHMMList.indexCnToEn();

fHMMList.indexFullEnToCn();

fHMMList.indexFullCnToEn();

fHMMList.indexFullCnToPy();

fHMMList.indexFullCnToKo();;

fHMMList.indexFullCnToJp();;

fHMMList.indexFullCnToTt();

fHMMList.indexFullCnToRs();

fHMMList.indexFullCnToAb();

neroController= new NERO\_C\_OneTime\_E();

nlpController= new Nlp\_CE\_XCDX\_S();

posController= new Pos\_CE\_XCDX\_P();

quick6DLuoYaoguangSort= new Quick6DLuoYaoguangSort3DMap\_E();

forestRoots= fHMMList.getMap();

forestsRoots= fHMMList.getMaps();

wordsForest= fHMMList.getPosCnToCn();

wordsForests= fHMMList.getWordsForests();

emotionMap= new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_NegativeMap();

emotionMap.IV\_PositiveMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

sensingTest= new SensingTest();

}

public List<String> parserMixedString(String mixedString) {

mixedString+= StablePOS.SPACE\_STRING\_DISTINCTION;

int inputStringLength= mixedString.length();

List<String> outputList = new LinkedList<>();

int forestDepth = StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords = new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO] = new StringBuilder();

fixWords[StablePOS.INT\_ONE] = new StringBuilder();

StringBuilder stringBuilder = new StringBuilder();

int find = StablePOS.INT\_ZERO;

Here:

for (int charPosition = StablePOS.INT\_ZERO;charPosition<inputStringLength;charPosition

+=(countInputStringLength==StablePOS.INT\_ZERO?StablePOS.INT\_ONE:countInputStringLength)) {if(mixedString.charAt(charPosition) < StablePOS.INT\_TEN\_SOUTHANDS && charPosition < inputStringLength - StablePOS.INT\_ONE){

if(find == StablePOS.INT\_ZERO) {

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

fixWords[StablePOS.INT\_ZERO].append(mixedString.charAt(charPosition));

countInputStringLength = StablePOS.INT\_ONE;

find = StablePOS.INT\_ONE;

continue Here;

}

if(StablePOS.INT\_ONE == find) {

find = StablePOS.INT\_ZERO;

Iterator<String> it = fHMMList.englishStringToWordsList(fixWords[StablePOS.INT\_ZERO].toString()).iterator();

StringBuilder number= new StringBuilder();

while(it.hasNext()) {

String temp = it.next();

if(StablePOS.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[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder = neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(mixedString

.charAt(charPosition)), mixedString, charPosition, inputStringLength, forestRoots, forestDepth, charPosition + StablePOS.INT\_ONE);

String countWordNode = stringBuilder.toString();

int compare = countInputStringLength = countWordNode.length();

if (StablePOS.INT\_ONE == compare) {

outputList.add(countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (StablePOS.INT\_TWO == compare) {

countInputStringLength = nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList

, stringBuilder, wordsForest, fixWords, posController, charPosition, mixedString);

continue Here;

}

if (StablePOS.INT\_THREE == compare) {

I\_FixWords(charPosition, mixedString, fixWords);

countInputStringLength = nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController, charPosition, mixedString);

continue Here;

}

if (StablePOS.INT\_FOUR == compare) {

I\_FixWords(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= StablePOS.INT\_ZERO;

int countInputStringLength;

StringBuilder[] fixWords= new StringBuilder[StablePOS.INT\_TWO];

fixWords[StablePOS.INT\_ZERO]= new StringBuilder();

fixWords[StablePOS.INT\_ONE]= new StringBuilder();

StringBuilder stringBuilder= new StringBuilder();

int find= StablePOS.INT\_ZERO;

Here:

for (int charPosition= StablePOS.INT\_ZERO; charPosition< inputStringLength; charPosition+= (countInputStringLength!= StablePOS.INT\_ZERO? countInputStringLength: StablePOS.INT\_ONE)) {

if(StablePOS.INT\_ONE\_TWO\_EIGHT> inputString.charAt(charPosition)){

if(fixWords[StablePOS.INT\_ZERO].length()> StablePOS.INT\_ZERO) {

if(fixWords[StablePOS.INT\_ZERO].charAt(fixWords[StablePOS.INT\_ZERO].length()- StablePOS.INT\_ONE)< StablePOS.INT\_ONE\_TWO\_EIGHT) { fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

find= StablePOS.INT\_ONE;

continue Here;

}

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

}

find= StablePOS.INT\_ONE; fixWords[StablePOS.INT\_ZERO].append(inputString.charAt(charPosition));

countInputStringLength= StablePOS.INT\_ONE;

continue Here;

}

if(find== StablePOS.INT\_ONE) {

find= StablePOS.INT\_ZERO;

outputList.add(fixWords[StablePOS.INT\_ZERO].toString());

}

stringBuilder.delete(StablePOS.INT\_ZERO, stringBuilder.length());

stringBuilder= neroController.getBinaryForestRecurWordOneTime(stringBuilder.append(inputString

.charAt(charPosition)), inputString, charPosition, inputStringLength, forestRoots, forestDepth, charPosition+ StablePOS.INT\_ONE);

String countWordNode= stringBuilder.toString();

int compare= countInputStringLength= countWordNode.length();

if (compare== StablePOS.INT\_ONE) {

outputList.add(countWordNode);

fixWords[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ZERO].length());

fixWords[StablePOS.INT\_ZERO].append(countWordNode);

continue Here;

}

if (compare== StablePOS.INT\_TWO) {

countInputStringLength= nlpController.doSlangPartAndPOSCheckForTwoChar(countInputStringLength, outputList

, stringBuilder, wordsForest, fixWords, posController, charPosition, inputString);

continue Here;

}

if (compare== StablePOS.INT\_THREE) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength= nlpController.doPOSAndEMMCheckOfThree(countInputStringLength, outputList

, wordsForest, stringBuilder, fixWords, posController, charPosition, inputString);

continue Here;

}

if (compare== StablePOS.INT\_FOUR) {

I\_FixWords(charPosition, inputString, fixWords);

countInputStringLength= nlpController.doSlangCheck(countInputStringLength, outputList, stringBuilder

, wordsForest, fixWords, posController, charPosition, inputString);

}

}

return outputList;

}

public void I\_FixWords(int charPosition, String inputString, StringBuilder[] fixWords) {

fixWords[StablePOS.INT\_ONE].delete(StablePOS.INT\_ZERO, fixWords[StablePOS.INT\_ONE].length());

if (charPosition + StablePOS.INT\_EIGHT < inputString.length()) {

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_THREE, charPosition + StablePOS.INT\_EIGHT));

return;

}

fixWords[StablePOS.INT\_ONE].append(inputString.substring(charPosition + StablePOS.INT\_THREE, inputString.length()));

}

public String[] parserEnglishString(String englishString) {

String[] words = englishString.replaceAll(StablePOS.NLP\_SPASE\_REP, StablePOS.SPACE\_STRING).split(StablePOS.SPACE\_STRING);

if(StablePOS.INT\_ZERO == words.length ) {

return new String[] {StablePOS.SPACE\_STRING};

}

return words;

}

}

WordFrequencyUtil, 词频类

package ME.utils;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

public class WordFrequencyUtil {

public static void WordFrequencyFindCheck(Map<String, WordFrequency> outputList,StringBuilder[] fixWords) {

String string= fixWords[StablePOS.INT\_ZERO].toString();

if (outputList.containsKey(string)) {

WordFrequency wordFrequency= outputList.get(string);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(string, wordFrequency);

return;

}

WordFrequency wordFrequency= new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(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.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(countWordNode, wordFrequency);

return;

}

WordFrequency wordFrequency=new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(fixWords[StablePOS.INT\_ZERO].toString());

outputList.put(countWordNode,wordFrequency);

}

}

WordForestUtil, 索引森林类

package ME.utils;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

public class WordForestUtil {

public static void wordsForestNotContainsKey(Map<String, WordFrequency> outputList

, String countWordNode, StringBuilder[] prefixWord) {

String string= String.valueOf(countWordNode.charAt(StablePOS.INT\_ZERO));

if (outputList.containsKey(string)) {

WordFrequency wordFrequency = outputList.get(string);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(string, wordFrequency);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode.charAt(StablePOS.INT\_ZERO));

return;

}

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(string);

outputList.put(string, wordFrequency);

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode.charAt(StablePOS.INT\_ZERO));

}

public static void prefixWordEqualZero(Map<String, WordFrequency> outputList,String countWordNode

, StringBuilder[] prefixWord) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

if (outputList.containsKey(countWordNode)) {

WordFrequency wordFrequency = outputList.get(countWordNode);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(countWordNode, wordFrequency);

return;

}

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(countWordNode);

outputList.put(countWordNode, wordFrequency);

}

public static void wordsForestContainsKey(Map<String, WordFrequency> outputList,String countWordNode

, StringBuilder[] prefixWord) {

prefixWord[StablePOS.INT\_ZERO].delete(StablePOS.INT\_ZERO, prefixWord[StablePOS.INT\_ZERO].length());

prefixWord[StablePOS.INT\_ZERO].append(countWordNode);

if (outputList.containsKey(countWordNode)) {

WordFrequency wordFrequency = outputList.get(countWordNode);

wordFrequency.I\_Frequency(wordFrequency.getFrequency() + StablePOS.INT\_ONE);

outputList.put(countWordNode, wordFrequency);

return;

}

WordFrequency wordFrequency = new WordFrequency();

wordFrequency.I\_Frequency(StablePOS.INT\_ONE);

wordFrequency.I\_Word(countWordNode);

outputList.put(countWordNode, wordFrequency);

}

}

SensingMapImp, 意识类

package OEI.AMV.ECS.SVQ.MPC.SOQ.OEM.E;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import OCI.AMV.ECS.SVQ.MPC.fhmm.C.SensingMap;

//import OCI.ME.analysis.C.A;

import OCI.ME.analysis.C.BinaryForest\_A;

public class SensingMap\_E implements SensingMap{

private Map<String, Object> lenovoMap;

@Override

public Map<String, Object> getLenovoMap() {

return this.lenovoMap;

}

@Override

public void I\_LenovoMap(Map<String, Object> lenovoMap) {

this.lenovoMap = lenovoMap;

}

@Override

public void IV\_LenovoMap(BinaryForest\_A \_A) throws IOException {

lenovoMap= new HashMap<>();

Map<String, String> CnToEnMap= \_A.getFullCnToEn();

Map<String, String> EnToCnMap= \_A.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)));

}

}

}

TranslatorImp, 翻译类

package MSU.OEI.ME.SMS.SEU.OSD.OSI.E;

import java.io.IOException;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.concurrent.CopyOnWriteArrayList;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.Verbal;

import SVQ.stable.StablePOS;

import MSU.OCI.ME.SMS.translator.C.Translator;

import OCI.ME.analysis.C.A;

public class Translator\_E 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(A \_A) throws IOException {

posec = \_A.getPosEnToCn();

posee = \_A.getPosEnToEn();

poscc = \_A.getPosCnToCn();

etc = \_A.getEnToCn();

cte = \_A.getCnToEn();

fulletc = \_A.getFullEnToCn();

fullcte = \_A.getFullCnToEn();

}

public String EnglishStringToChineseString(A \_A, String EnglishString) {

String[] nodes = \_A.parserEnglishString(EnglishString);

StringBuilder sb = new StringBuilder();

for(String temp:nodes) {

char[] caseTemp = temp.toCharArray();

caseTemp[StablePOS.INT\_ONE] = String.valueOf(caseTemp[StablePOS.INT\_ONE])

.toUpperCase().charAt(StablePOS.INT\_ONE);

if(etc.containsKey(temp)) {

sb.append(etc.get(temp).split(StablePOS.NLP\_DOT)[StablePOS.INT\_ONE]);

}else if(fulletc.containsKey(String.valueOf(caseTemp))){

sb.append(fulletc.get(String.valueOf(caseTemp)));

}else {

sb.append(temp);

}

sb.append(StablePOS.SPACE\_STRING);

}

return sb.toString();

}

public String ChineseStringToEnglishString(A \_A, String ChineseString) {

List<String> nodes = \_A.parserMixedString(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(StablePOS.SPACE\_STRING);

}

return sb.toString();

}

public String MixedStringToChineseString(A \_A, String mixedString) {

List<String> nodes = \_A.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 = \_A.parserEnglishString(temp);

for(String string:strings) {

if(string == null || string.length() < StablePOS.INT\_ONE) {

string = StablePOS.EMPTY\_STRING;

}

if(etc.containsKey(string)) {

sb.append(etc.get(string).split(StablePOS.NLP\_DOT)[StablePOS.INT\_ZERO]);

}else if(fulletc.containsKey(string)){

sb.append(fulletc.get(string));

}else {

sb.append(temp);

}

}

}

}

return sb.toString();

}

public String ChineseStringToEnglishStringWithPOS(A \_A, String ChineseString) {

List<String> nodes = \_A.parserMixedString(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(StablePOS.NLP\_SYMBO\_SLASH);

sb.append(poscc.get(temp));

}

sb.append(StablePOS.SPACE\_STRING);

}

return sb.toString();

}

public List<Verbal> index(A \_A, String mixedString) {

List<Verbal> verbals = new CopyOnWriteArrayList<>();

List<String> nodes = \_A.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(StablePOS.NLP\_NULL);

verbal.setExplain(StablePOS.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(StablePOS.NLP\_ZI\_DONG)){

if(!verbals.get(i).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

if(i - StablePOS.INT\_ONE > StablePOS.INT\_ZERO && verbals.get(i - StablePOS.INT\_ONE)

.getEnglish().contains(StablePOS.NLP\_ENGLISH\_OF)) {

Verbal temp = verbals.get(i);

temp.setPartOfSpeech(StablePOS.NLP\_CI\_DONG\_MING);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length() - StablePOS.INT\_ONE) == StablePOS.NLP\_CHAR\_E) {

english = StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING

+ english.substring(StablePOS.INT\_ZERO, english.length() - StablePOS.INT\_ONE)

+ StablePOS.NLP\_ENGLISH\_ING;}else {

english = StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING + english + StablePOS.NLP\_ENGLISH\_ING;

}

temp.setEnglish(english);

}else if(verbals.get(i + StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

if(i - StablePOS.INT\_ONE >= StablePOS.INT\_ZERO && !verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

if(!verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_DAI)){

if(verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_JIE)){

if(verbals.get(i - StablePOS.INT\_ONE).getChinese().contains(StablePOS.NLP\_ZI\_ZAI)){

if(verbals.get(i + StablePOS.INT\_ONE).getChinese().contains(StablePOS.NLP\_ZI\_ZHONG)){

Verbal temp = verbals.get(i + StablePOS.INT\_ONE);

temp.setEnglish(StablePOS.NLP\_ENGLISH\_STATUS);

}

}

}

Verbal temp = verbals.get(i);

temp.setPartOfSpeech(StablePOS.NLP\_CI\_DONG\_MING);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length()-StablePOS.INT\_ONE) == StablePOS.NLP\_CHAR\_E) {

english = StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING

+ english.substring(StablePOS.INT\_ZERO, english.length() - StablePOS.INT\_ONE)

+ StablePOS.NLP\_ENGLISH\_ING;}else {

english = StablePOS.NLP\_ENGLISH\_THE

+ StablePOS.SPACE\_STRING + english + StablePOS.NLP\_ENGLISH\_ING;

}

temp.setEnglish(english);

}

}

if(verbals.get(i + StablePOS.INT\_TWO).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_DONG)){

if(!verbals.get(i + StablePOS.INT\_TWO).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

if(verbals.get(i + StablePOS.INT\_THREE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

Verbal temp = verbals.get(i + StablePOS.INT\_TWO);

temp.setPartOfSpeech(StablePOS.NLP\_CI\_DONG\_MING);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length() - StablePOS.INT\_ONE) == StablePOS.NLP\_CHAR\_E) {

english = StablePOS.NLP\_ENGLISH\_OF + StablePOS.SPACE\_STRING +

StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING + english.substring(StablePOS.INT\_ZERO, english.length()-StablePOS.INT\_ONE)

+ StablePOS.NLP\_ENGLISH\_ING; }else {

english = StablePOS.NLP\_ENGLISH\_OF + StablePOS.SPACE\_STRING +

StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING + english + StablePOS.NLP\_ENGLISH\_ING;

}

temp.setEnglish(english);

}

}

}

}else if(i - StablePOS.INT\_ONE >= StablePOS.INT\_ZERO && verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_JIE)){

Verbal temp = verbals.get(i);

temp.setPartOfSpeech(StablePOS.NLP\_CI\_DONG\_MING);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length() - StablePOS.INT\_ONE) == StablePOS.NLP\_CHAR\_E) {

english = StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING + english.substring(StablePOS.INT\_ZERO, english.length() - StablePOS.INT\_ONE) + StablePOS.NLP\_ENGLISH\_ING;}else {english = StablePOS.NLP\_ENGLISH\_THE + StablePOS.SPACE\_STRING + english + StablePOS.NLP\_ENGLISH\_ING;

}temp.setEnglish(english);

}else if(i - StablePOS.INT\_ONE >= StablePOS.INT\_ZERO && verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_FU)){

Verbal temp = verbals.get(i);

temp.setPartOfSpeech(StablePOS.NLP\_CI\_DONG\_MING);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length() - StablePOS.INT\_ONE) == StablePOS.NLP\_CHAR\_E) {

english = english.substring(StablePOS.INT\_ZERO, english.length() - StablePOS.INT\_ONE)

+ StablePOS.NLP\_ENGLISH\_ING; }else {

english += StablePOS.NLP\_ENGLISH\_ING;

}

temp.setEnglish(english);

}

}

}else if(verbals.get(i).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING)){

if(i - StablePOS.INT\_ONE >= StablePOS.INT\_ZERO && verbals.get(i - StablePOS.INT\_ONE)

.getPartOfSpeech().contains(StablePOS.NLP\_FU\_SHU)){

Verbal temp = verbals.get(i);

String english = temp.getEnglish().replace(StablePOS.SPACE\_STRING, StablePOS.EMPTY\_STRING);

if(english.charAt(english.length()-1)

== StablePOS.NLP\_CHAR\_H || english.charAt(english.length()-1)

== StablePOS.NLP\_CHAR\_S) {

english += StablePOS.NLP\_ENGLISH\_ES;

}else {

english += StablePOS.NLP\_ENGLISH\_S;

}

temp.setEnglish(english);

}

}else if(verbals.get(i).getPartOfSpeech().contains(StablePOS.NLP\_CI\_FU)){

if(i - StablePOS.INT\_ONE >= StablePOS.INT\_ZERO && (verbals.get(i-StablePOS.INT\_ONE)

.getPartOfSpeech().contains(StablePOS.NLP\_ZI\_DAI)

||verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_MING))){

if(i-StablePOS.INT\_ONE >= StablePOS.INT\_ZERO&&verbals.get(i - StablePOS.INT\_ONE).getPartOfSpeech()

.contains(StablePOS.NLP\_ZI\_复)

||(verbals.get(i-StablePOS.INT\_ONE).getPartOfSpeech()

.contains(StablePOS.NLP\_ZI\_单)

&& verbals.get(i-StablePOS.INT\_ONE).getPartOfSpeech().contains(StablePOS.NLP\_ZI\_一))){

if(verbals.get(i).getEnglish().contains(StablePOS.NLP\_HAVE\_HAS)) {

Verbal temp = verbals.get(i);

String english = temp.getEnglish().replace(StablePOS.NLP\_HAVE\_HAS, StablePOS.NLP\_HAVE);

temp.setEnglish(english);

}

}else {

if(verbals.get(i).getEnglish().contains(StablePOS.NLP\_HAVE\_HAS)) {

Verbal temp = verbals.get(i);

String english = temp.getEnglish().replace(StablePOS.NLP\_HAVE\_HAS, StablePOS.NLP\_HAS);

temp.setEnglish(english);

}

}

}

if(i-StablePOS.INT\_TWO>= StablePOS.INT\_ZERO && (verbals.get(i-StablePOS.INT\_TWO).getPartOfSpeech()

.contains(StablePOS.NLP\_ZI\_DAI) || verbals.get(i-StablePOS.INT\_TWO).getPartOfSpeech()

.contains(StablePOS.NLP\_ZI\_MING))){

if(verbals.get(i-StablePOS.INT\_TWO)

.getPartOfSpeech().contains(StablePOS.NLP\_ZI\_复)

||(verbals.get(i-StablePOS.INT\_TWO)

.getPartOfSpeech().contains(StablePOS.NLP\_ZI\_单)

&&verbals.get(i-StablePOS.INT\_TWO)

.getPartOfSpeech().contains(StablePOS.NLP\_ZI\_一))){

if(verbals.get(i).getEnglish().contains(StablePOS.NLP\_HAVE\_HAS)) {

Verbal temp = verbals.get(i);

String english = temp.getEnglish().replace(StablePOS.NLP\_HAVE\_HAS, StablePOS.NLP\_HAVE);

temp.setEnglish(english);

}

}else {

if(verbals.get(i).getEnglish().contains(StablePOS.NLP\_HAVE\_HAS)) {

Verbal temp = verbals.get(i);

String english = temp.getEnglish().replace(StablePOS.NLP\_HAVE\_HAS, StablePOS.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 ? StablePOS.EMPTY\_STRING : verbals.get(i).getChinese());

}

return sb.toString().replaceAll(StablePOS.NLP\_SPASE\_REP, StablePOS.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 ? StablePOS.EMPTY\_STRING : verbalsFix.get(i).getEnglish().toLowerCase());

sb.append(StablePOS.SPACE\_STRING);

}

return sb.toString().replaceAll(StablePOS.NLP\_SPASE\_REP, StablePOS.SPACE\_STRING);

}

}

LenovoTest, 联想类

package OSU.PEQ.AVC.SUQ.test;

import java.io.IOException;

import java.util.Iterator;

import java.util.Map;

import AEU.AVC.SUQ.engine.LenovoInit;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

public class LenovoTest{

public static void main(String[] argv) throws IOException {

//init

String text = "..请用户 自行添加sample.";

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()+" ");

}

}

}

}

}

EnvironmentTest, 环境类

public class EnvironmentTest{

public static void main(String[] argv) throws IOException {

//init

String text = "";

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()+" ");

}

}

}

}

EmotionTest, 思维类

package OSU.PEQ.AVC.SUQ.test;

import java.io.IOException;

import AEU.AVC.SUQ.engine.EmotionInit;

public class EmotionTest{

public static void main(String[] argv) throws IOException {

String text = "";

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);

}

}

EnvironmentInit, 环境类

package OSU.PEQ.AVC.SUQ.test;

import java.io.IOException;

import AEU.AVC.SUQ.engine.EmotionInit;

package AEU.AVC.SUQ.engine;

import java.io.IOException;

import java.util.List;

import java.util.Map;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import AEU.OCI.AVC.SUQ.estimation.C.RatioMap;

import AEU.OEI.AVC.SUQ.SVU.EOP.E.RatioMap\_E;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A;

//import OCI.ME.analysis.C.A;

import OEI.AVC.SUQ.SVQ.MPC.fhmm.E.EmotionMap\_E;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

import SVQ.stable.StableString;

public class EnvironmentInit{

public EmotionMap getEmotionMap() {

return emotionMap;

}

public void I\_EmotionMap(EmotionMap emotionMap) {

this.emotionMap = emotionMap;

}

public A get\_A() {

return \_A;

}

public void I\_\_A(CogsBinaryForest\_AE \_A) {

this.\_A = \_A;

}

public Map<String, Object> getPositive() {

return positive;

}

public void I\_Positive(Map<String, Object> positive) {

this.positive = positive;

}

public Map<String, Object> getNegative() {

return negative;

}

public void I\_Negative(Map<String, Object> negative) {

this.negative = negative;

}

public Map<String, Object> getMotivation() {

return motivation;

}

public void I\_Motivation(Map<String, Object> motivation) {

this.motivation = motivation;

}

public Map<String, Object> getTrending() {

return trending;

}

public void I\_Trending(Map<String, Object> trending) {

this.trending = trending;

}

public Map<String, Object> getPrediction() {

return prediction;

}

public void I\_Prediction(Map<String, Object> prediction) {

this.prediction = prediction;

}

public List<String> getSets() {

return sets;

}

public void I\_Sets(List<String> sets) {

this.sets = sets;

}

public RatioMap getRationMap() {

return rationMap;

}

public void I\_RationMap(RatioMap rationMap) {

this.rationMap = rationMap;

}

public Map<Integer, WordFrequency> getWordFrequencyMap() {

return wordFrequencyMap;

}

public void I\_WordFrequencyMap(Map<Integer, WordFrequency> wordFrequencyMap) {

this.wordFrequencyMap = wordFrequencyMap;

}

public Map<String, EmotionSample> getEmotionSampleMap() {

return emotionSampleMap;

}

public void I\_EmotionSampleMap(Map<String, EmotionSample> emotionSampleMap) {

this.emotionSampleMap = emotionSampleMap;

}

public double getPositiveCount() {

return positiveCount;

}

public void I\_PositiveCount(double positiveCount) {

this.positiveCount = positiveCount;

}

public double getNegativeCount() {

return negativeCount;

}

public void I\_NegativeCount(double negativeCount) {

this.negativeCount = negativeCount;

}

public double getTotalCount() {

return totalCount;

}

public void I\_TotalCount(double totalCount) {

this.totalCount = totalCount;

}

public Map<String, Object> getDistinction() {

return distinction;

}

public void I\_Distinction(Map<String, Object> distinction) {

this.distinction = distinction;

}

private EmotionMap emotionMap;

private CogsBinaryForest\_AE \_A;

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 {

EnvironmentInit environmentInit = new EnvironmentInit();

environmentInit.IV\_(StableString.text1);

Map<String, EmotionSample> environmentSampleMap = environmentInit.getEmotionSampleMap();

}

public void IV\_(String text) throws IOException {

emotionMap = new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

emotionMap.IV\_DistinctionMap();

\_A = new CogsBinaryForest\_AE();

//\_A.IV\_Mixed();

\_A.IV\_Mixed();

motivation = emotionMap.getMotivationMap();

trending = emotionMap.getTrendingMap();

prediction = emotionMap.getPredictionMap();

distinction = emotionMap.getDistinctionMap();

//sets = \_A.parserMixedString(text);//稍后统一更新下接口为parserMixedString

sets = \_A.parserMixedString(text);//

wordFrequencyMap = \_A.getWordFrequencyByReturnSortMap(sets);

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEnvironmentSampleMap(wordFrequencyMap);

rationMap.IQ\_Motivation(emotionSampleMap, motivation);

rationMap.IQ\_Trending(emotionSampleMap, trending);

rationMap.IQ\_Prediction(emotionSampleMap, prediction);

rationMap.IQ\_Distinction(emotionSampleMap, distinction);

}

public void IV\_Exclude\_A(String text, CogsBinaryForest\_AE \_AInput) throws IOException {

emotionMap = new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

emotionMap.IV\_DistinctionMap();

\_A = \_AInput;

motivation = emotionMap.getMotivationMap();

trending = emotionMap.getTrendingMap();

prediction = emotionMap.getPredictionMap();

distinction = emotionMap.getDistinctionMap();

sets = \_AInput.parserString(text);

wordFrequencyMap = \_AInput.getWordFrequencyByReturnSortMap(sets);

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEnvironmentSampleMap(wordFrequencyMap);

rationMap.IQ\_Motivation(emotionSampleMap, motivation);

rationMap.IQ\_Trending(emotionSampleMap, trending);

rationMap.IQ\_Prediction(emotionSampleMap, prediction);

rationMap.IQ\_Distinction(emotionSampleMap, distinction);

}

public void IV\_FromEmotion(Map<Integer, WordFrequency> getWordFrequencyMap) throws IOException {

emotionMap = new EmotionMap\_E();

emotionMap.IV\_MotivationMap();

emotionMap.IV\_TrendingMap();

emotionMap.IV\_PredictionMap();

emotionMap.IV\_DistinctionMap();

//parser sentence

motivation = emotionMap.getMotivationMap();

trending = emotionMap.getTrendingMap();

prediction = emotionMap.getPredictionMap();

distinction = emotionMap.getDistinctionMap();

//map

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEnvironmentSampleMap(getWordFrequencyMap);

rationMap.IQ\_Motivation(emotionSampleMap, motivation);

rationMap.IQ\_Trending(emotionSampleMap, trending);

rationMap.IQ\_Prediction(emotionSampleMap, prediction);

rationMap.IQ\_Distinction(emotionSampleMap, distinction);

}

public void IV\_FromEmotionExcludeEmotion(Map<Integer, WordFrequency> getWordFrequencyMap, EmotionMap emotionMapInput) throws IOException {

emotionMap = emotionMapInput;

motivation = emotionMap.getMotivationMap();

trending = emotionMap.getTrendingMap();

prediction = emotionMap.getPredictionMap();

distinction = emotionMap.getDistinctionMap();

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEnvironmentSampleMap(getWordFrequencyMap);

rationMap.IQ\_Motivation(emotionSampleMap, motivation);

rationMap.IQ\_Trending(emotionSampleMap, trending);

rationMap.IQ\_Prediction(emotionSampleMap, prediction);

rationMap.IQ\_Distinction(emotionSampleMap, distinction);

}

}

EmotionInit, 思维类

public class EmotionInit{

public EmotionMap getEmotionMap() {

return emotionMap;

}

public void I\_EmotionMap(EmotionMap emotionMap) {

this.emotionMap = emotionMap;

}

//稍后进行 A 新陈代谢分化

//20210702

public A get\_A() {

return \_A;

}

public void I\_\_A(CogsBinaryForest\_AE \_A) {

this.\_A = \_A;

}

public Map<String, Object> getPositive() {

return positive;

}

public void I\_Positive(Map<String, Object> positive) {

this.positive = positive;

}

public Map<String, Object> getNegative() {

return negative;

}

public void I\_Negative(Map<String, Object> negative) {

this.negative = negative;

}

public Map<String, Object> getMotivation() {

return motivation;

}

public void I\_Motivation(Map<String, Object> motivation) {

this.motivation = motivation;

}

public Map<String, Object> getTrending() {

return trending;

}

public void I\_Trending(Map<String, Object> trending) {

this.trending = trending;

}

public Map<String, Object> getPrediction() {

return prediction;

}

public void I\_Prediction(Map<String, Object> prediction) {

this.prediction = prediction;

}

public List<String> getSets() {

return sets;

}

public void I\_Sets(List<String> sets) {

this.sets = sets;

}

public RatioMap getRationMap() {

return rationMap;

}

public void I\_RationMap(RatioMap rationMap) {

this.rationMap = rationMap;

}

public Map<Integer, WordFrequency> getWordFrequencyMap() {

return wordFrequencyMap;

}

public void I\_WordFrequencyMap(Map<Integer, WordFrequency> wordFrequencyMap) {

this.wordFrequencyMap = wordFrequencyMap;

}

public Map<String, EmotionSample> getEmotionSampleMap() {

return emotionSampleMap;

}

public void I\_EmotionSampleMap(Map<String, EmotionSample> emotionSampleMap) {

this.emotionSampleMap = emotionSampleMap;

}

public double getPositiveCount() {

return positiveCount;

}

public void I\_PositiveCount(double positiveCount) {

this.positiveCount = positiveCount;

}

public double getNegativeCount() {

return negativeCount;

}

public void I\_NegativeCount(double negativeCount) {

this.negativeCount = negativeCount;

}

public double getTotalCount() {

return totalCount;

}

public void I\_TotalCount(double totalCount) {

this.totalCount = totalCount;

}

private EmotionMap emotionMap;

private CogsBinaryForest\_AE \_A;

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 {

EmotionInit emotionInit = new EmotionInit();

emotionInit.IV\_(StableString.text1);

}

public void IV\_(String text) throws IOException {

emotionMap = new EmotionMap\_E();

emotionMap.IV\_NegativeMap();

emotionMap.IV\_PositiveMap();

\_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

positive = emotionMap.getPositiveMap();

negative = emotionMap.getNegativeMap();

sets = \_A.parserMixedString(text);

wordFrequencyMap = \_A.getWordFrequencyByReturnSortMap(sets);

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);

positiveCount = rationMap.findTotalPositiveCount(emotionSampleMap);

negativeCount = rationMap.findTotalNegativeCount(emotionSampleMap);

totalCount = rationMap.findTotalKeyCount(emotionSampleMap);

}

public void IV\_Exclude\_A(String text, CogsBinaryForest\_AE \_AInput, EmotionMap emotionMapInput) throws IOException {

emotionMap = emotionMapInput;

\_A = \_AInput;

positive = emotionMap.getPositiveMap();

negative = emotionMap.getNegativeMap();

sets = \_A.parserMixedString(text);

wordFrequencyMap = \_A.getWordFrequencyByReturnSortMap(sets);

rationMap = new RatioMap\_E();

emotionSampleMap = rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);

positiveCount = rationMap.findTotalPositiveCount(emotionSampleMap);

negativeCount = rationMap.findTotalNegativeCount(emotionSampleMap);

totalCount = rationMap.findTotalKeyCount(emotionSampleMap);

}

}

LenovoInit, 联想类

package AEU.AVC.SUQ.engine;

import java.io.IOException;

import java.util.Map;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import OCI.AMV.ECS.SVQ.MPC.fhmm.C.SensingMap;

import OCI.ME.analysis.C.A;

import OEI.AMV.ECS.SVQ.MPC.SOQ.OEM.E.SensingMap\_E;

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 = "";

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 SensingMap\_E();

sensingMap.initLenovoMap(environmentInit.getAnalyzer());

}

public void initExcludeAnalyzer(String text, A \_A) throws IOException {

environmentInit = new EnvironmentInit();

environmentInit.initExcludeAnalyzer(text, \_A);

sensingMap = new SensingMap\_E();

sensingMap.initLenovoMap(environmentInit.getAnalyzer());

}

}

RatioMapImp, 比率图类

package AEU.OEI.AVC.SUQ.SVU.EOP.E;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import AEU.OCI.AVC.SUQ.estimation.C.RatioMap;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

public class RatioMap\_E 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() - StablePOS.INT\_ONE; i >= StablePOS.INT\_ZERO; i--) {

if(wordFrequencyMap.get(i).getWord().length() > StablePOS.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.I\_PositiveCount(wordFrequencyMap.get(i).getFrequency());

}else if(negative.containsKey(wordFrequencyMap.get(i).getWord())) {

emotionSample.I\_NegativeCount(wordFrequencyMap.get(i).getFrequency());

}else {

emotionSample.I\_MedCount(wordFrequencyMap.get(i).getFrequency());

}

output.put(wordFrequencyMap.get(i).getWord(), emotionSample);

}

}

return output;

}

@Override

public void IQ\_MotivationRatio(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.I\_MotivationRatio(emotionSample.getEmotionRatio()/sumOfEmotion);

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_CorrelationRatio(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.I\_CorrelationRatio((emotionSample.getPositiveCount()

+ emotionSample.getNegativeCount() + emotionSample.getMedCount())/sumOfEmotion);

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_ContinusRatio(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.I\_ContinusRatio((emotionSample.getPositiveCount()

+ emotionSample.getNegativeCount() + emotionSample.getMedCount()) \* emotionRatio);

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_TrendsRatio(Map<String, EmotionSample> emotionSampleMap) {

Iterator<String> Iterator = emotionSampleMap.keySet().iterator();

while(Iterator.hasNext()) {

String word = Iterator.next();

EmotionSample emotionSample = emotionSampleMap.get(word);

emotionSample.I\_TrendsRatio(emotionSample.getEmotionRatio() \* emotionSample.getContinusRatio()\* emotionSample.getCorrelationRatio());

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_PredictionRatio(Map<String, EmotionSample> emotionSampleMap) {

Iterator<String> Iterator = emotionSampleMap.keySet().iterator();

while(Iterator.hasNext()) {

String word = Iterator.next();

EmotionSample emotionSample = emotionSampleMap.get(word);

emotionSample.I\_PredictionRatio(emotionSample.getMotivationRatio()\*emotionSample.getCorrelationRatio());

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_GuessRatio(Map<String, EmotionSample> emotionSampleMap) {

Iterator<String> Iterator = emotionSampleMap.keySet().iterator();

while(Iterator.hasNext()) {

String word = Iterator.next();

EmotionSample emotionSample = emotionSampleMap.get(word);

emotionSample.I\_GuessRatio(emotionSample.getPredictionRatio()\*emotionSample.getTrendsRatio());

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_SensingRatio(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.I\_SensingRatio(0);

}else {

emotionSample.I\_SensingRatio(emotionSample.getPredictionRatio()/emotionSample.getTrendsRatio());

}

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public double findTotalPositiveCount(Map<String, EmotionSample> emotionSampleMap) {

double output = StablePOS.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 = StablePOS.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 IQ\_EmotionRatio(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.I\_EmotionRatio(negRate + posRate + medRate);

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public double findTotalKeyCount(Map<String, EmotionSample> emotionSampleMap) {

double output = StablePOS.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 IQ\_Motivation(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.I\_Motivation(motivation.get(word).toString());

}

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_Trending(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.I\_Trending(trending.get(word).toString());

}

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_Prediction(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.I\_Prediction(prediction.get(emotionSample.getTrending()).toString());

} else if(prediction.containsKey(emotionSample.getMotivation())) {

emotionSample.I\_Prediction(prediction.get(emotionSample.getMotivation()).toString());

}

emotionSampleMap.put(word, emotionSample);

}

}

@Override

public void IQ\_Distinction(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.I\_Distinction(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() - StablePOS.INT\_ONE;

i >= StablePOS.INT\_ZERO; i--) {

if(wordFrequencyMap.get(i).getWord().length() > StablePOS.INT\_ONE) {

EmotionSample emotionSample = new EmotionSample();

if(!output.containsKey(wordFrequencyMap.get(i).getWord())) {

output.put(wordFrequencyMap.get(i).getWord(), emotionSample);

}

}

}

return output;

}

}

SuccessICATest, 比率分析类

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 = "";

String text2 = "";

String text3 = "";

//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

}

}

LiterarinessLevelTest, 文学类

public class LiterarinessLevelTest{

double[] literarinessLevel;

double literarinessDuration;

public void getEducationLevel(double[][] measurement) {

literarinessLevel= new double[measurement.length];

for(int i= StablePOS.INT\_ZERO; i< measurement.length; i++) {

literarinessLevel[i]=(measurement[i][StablePOS.INT\_ZERO]

\*measurement[i][StablePOS.INT\_THREE])/measurement[i][StablePOS.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 = "";

String text2 = "";

String text3 = "";

//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);

}

}

EducationLevelTest, 教育类

package PEQ.OPM.VEC.test;

import java.io.IOException;

import AEU.OPM.VEC.ica.EducationRatio;

import SVQ.stable.StablePOS;

public class EducationLevelTest{

double[] EducationLevel;

double EducationDuration;

public void getEducationLevel(double[][] measurement) {

EducationLevel = new double[measurement.length];

for(int i = StablePOS.INT\_ZERO; i < measurement.length; i++) {

EducationLevel[i] = (measurement[i][StablePOS.INT\_THREE] + measurement[i][StablePOS.INT\_FOUR])/(measurement[i][StablePOS.INT\_ONE] + measurement[i][StablePOS.INT\_TWO] + measurement[i][StablePOS.INT\_THREE] + measurement[i][StablePOS.INT\_FOUR]+ measurement[i][StablePOS.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 = "";

String text2 = "";

String text3 = "";

//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);

}

}

InitBehaviorICAKernel, 习惯类

package AEU.OPM.VEC.ica;

import java.io.IOException;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import AEU.AVC.SUQ.engine.EmotionInit;

import AEU.AVC.SUQ.engine.EnvironmentInit;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import SVQ.stable.StablePOS;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A;

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 += StablePOS.INT\_ONE;

return positiveCountEnvironment/totalCountEnvironment;

}

public double getTrustRate(String text, A \_A, EmotionMap emotionMap) throws IOException {

EmotionInit emotionInitEnvironment = new EmotionInit();

emotionInitEnvironment.initExcludeAnalyzer(text, \_A, emotionMap);

//reduce

double positiveCountEnvironment = emotionInitEnvironment.getPositiveCount();

double totalCountEnvironment = emotionInitEnvironment.getTotalCount();

positiveCountEnvironment += StablePOS.INT\_ONE;

return positiveCountEnvironment/totalCountEnvironment;

}

public double[] getBehaviorICAKernel(String text) throws IOException {

forRestReturn = new LinkedList<>();

kernel = new double[StablePOS.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 == StablePOS.INT\_ZERO) {

positiveCount = StablePOS.INT\_ONE;

}

if(negativeCount == StablePOS.INT\_ZERO) {

negativeCount = StablePOS.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[StablePOS.INT\_ZERO] = adjRatio;

kernel[StablePOS.INT\_ONE] = phychologicRatio;

kernel[StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(environmentText);

kernel[StablePOS.INT\_THREE] = getTrustRate(environmentText);

forRestReturn.add(StablePOS.EMPTY\_STRING + kernel[StablePOS.INT\_THREE]);

forRestReturn.add("信任比率：");

String motivationText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(motivationText);

kernel[StablePOS.INT\_FOUR] = getTrustRate(motivationText);

forRestReturn.add(StablePOS.EMPTY\_STRING+kernel[StablePOS.INT\_FOUR]);

forRestReturn.add("执行比率：");

String trendingText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(trendingText);

kernel[StablePOS.INT\_FIVE] = getTrustRate(trendingText);

forRestReturn.add(StablePOS.EMPTY\_STRING + kernel[StablePOS.INT\_FIVE]);

forRestReturn.add("成功比率：");

String predictionText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(predictionText);

kernel[StablePOS.INT\_SIX] = getTrustRate(predictionText);

forRestReturn.add(StablePOS.EMPTY\_STRING + kernel[StablePOS.INT\_SIX]);

return kernel;

}

public double[] getBehaviorICAKernel(String text, A \_A, EmotionMap emotionMap) throws IOException {

forRestReturn = new LinkedList<>();

kernel = new double[StablePOS.INT\_SEVEN];

EmotionInit emotionInit = new EmotionInit();

emotionInit.initExcludeAnalyzer(text, \_A, 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 == StablePOS.INT\_ZERO) {

positiveCount = StablePOS.INT\_ONE;

}

if(negativeCount == StablePOS.INT\_ZERO) {

negativeCount = StablePOS.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[StablePOS.INT\_ZERO] = adjRatio;

kernel[StablePOS.INT\_ONE] = phychologicRatio;

kernel[StablePOS.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 = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(environmentText);

kernel[StablePOS.INT\_THREE] = getTrustRate(environmentText, \_A, emotionMap);

forRestReturn.add(StablePOS.EMPTY\_STRING+kernel[StablePOS.INT\_THREE]);

forRestReturn.add("信任比率：");

String motivationText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(motivationText);

kernel[StablePOS.INT\_FOUR] = getTrustRate(motivationText, \_A, emotionMap);

forRestReturn.add(StablePOS.EMPTY\_STRING+kernel[StablePOS.INT\_FOUR]);

forRestReturn.add("执行比率：");

String trendingText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(trendingText);

kernel[StablePOS.INT\_FIVE] = getTrustRate(trendingText, \_A, emotionMap);

forRestReturn.add(StablePOS.EMPTY\_STRING+kernel[StablePOS.INT\_FIVE]);

forRestReturn.add("成功比率：");

String predictionText = StablePOS.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() + StablePOS.SPACE\_STRING;

}

}

forRestReturn.add(predictionText);

kernel[StablePOS.INT\_SIX] = getTrustRate(predictionText, \_A, emotionMap);

forRestReturn.add(StablePOS.EMPTY\_STRING+kernel[StablePOS.INT\_SIX]);

return kernel;

}

}

EducationRatio, 教育类

package AEU.OPM.VEC.ica;

import java.io.IOException;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import OCI.ME.analysis.C.A;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

public class EducationRatio{

public double[] getEducationKernel(String text) throws IOException {

A \_A = new CogsBinaryForest\_AE();

\_A.IV\_Mixed();

Map<String, String> pos = \_A.getPosCnToCn();

List<String> sets = \_A.parserMixedString(text);

Map<Integer, WordFrequency> map = \_A.getWordFrequencyByReturnSortMap(sets);

double[] output = new double[StablePOS.INT\_SEVEN];

output[StablePOS.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(StablePOS.NLP\_ZI\_MING)){

output[StablePOS.INT\_ONE]+= StablePOS.INT\_ONE;

continue Here;

}

if(pos.get(wordFrequency.getWord()).contains(StablePOS.NLP\_ZI\_DONG)){

output[StablePOS.INT\_TWO]+= StablePOS.INT\_ONE;

continue Here;

}

if(pos.get(wordFrequency.getWord()).contains(StablePOS.NLP\_ZI\_WEI)){

output[StablePOS.INT\_FOUR]+= StablePOS.INT\_ONE;

continue Here;

}

if(pos.get(wordFrequency.getWord()).contains(StablePOS.NLP\_ZI\_XING)){

output[StablePOS.INT\_THREE]+= StablePOS.INT\_ONE;

continue Here;

}

if(pos.get(wordFrequency.getWord()).contains(StablePOS.NLP\_ZI\_FU)){

output[StablePOS.INT\_FIVE]+= StablePOS.INT\_ONE;

continue Here;

}

if(pos.get(wordFrequency.getWord()).contains(StablePOS.NLP\_ZI\_JIE)){

output[StablePOS.INT\_SIX]+= StablePOS.INT\_ONE;

}

}

return output;

}

public double[] getEducationRatioKernel(double[] input) {

double[] output=new double[input.length];

for(int i=StablePOS.INT\_ZERO;i<input.length;i++) {

output[i]=input[i]/input[StablePOS.INT\_ZERO];

}

return output;

}

}

第四节 DNA元基索引版本等

Sensing等maptest的过程函数补充

package PEQ.AMV.ECS.test;

import java.io.IOException;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import AEU.OCI.AVC.SUQ.estimation.C.EmotionSample;

import AEU.OCI.AVC.SUQ.estimation.C.RatioMap;

import AEU.OEI.AVC.SUQ.SVU.EOP.E.RatioMap\_E;

import AVQ.ASQ.OVQ.OSQ.VSQ.obj.WordFrequency;

import SVQ.stable.StablePOS;

import OCI.AVC.SUQ.SVQ.MPC.fhmm.C.EmotionMap;

import OCI.ME.analysis.C.A;

import OEI.AVC.SUQ.SVQ.MPC.fhmm.E.EmotionMap\_E;

import OEI.ME.analysis.E.CogsBinaryForest\_AE;

public class SensingTest{

public List<String> getSets() {

return sets;

}

public void setSets(List<String> sets) {

this.sets = sets;

}

public Map<String, String> getPosCnToCn() {

return this.pos;

}

private List<String> sets;

private Map<String, String> pos;

public String[][] getMatrix() throws IOException {

EmotionMap emotionMap = new EmotionMap\_E();

emotionMap.initMotivationMap();

emotionMap.initNegativeMap();

emotionMap.initPositiveMap();

emotionMap.initTrendingMap();

emotionMap.initPredictionMap();

String text = " ";

A \_A = new CogsBinaryForest\_AE();

\_A.initMixed();

pos = \_A.getPosCnToCn();

Map<String, Object> positive= emotionMap.getPositiveMap();

Map<String, Object> negative= emotionMap.getNegativeMap();

Map<String, Object> motivation= emotionMap.getMotivationMap();

Map<String, Object> trending= emotionMap.getTrendingMap();

Map<String, Object> prediction= emotionMap.getPredictionMap();

sets = \_A.parserMixedString(text);

Map<Integer, WordFrequency> wordFrequencyMap= \_A.getWordFrequencyByReturnSortMap(sets);

RatioMap rationMap= new RatioMap\_E();

Map<String, EmotionSample> emotionSampleMap= rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);

double positiveCount= rationMap.findTotalPositiveCount(emotionSampleMap);

double negativeCount= rationMap.findTotalNegativeCount(emotionSampleMap);

double totalCount= rationMap.findTotalKeyCount(emotionSampleMap);

double medCount= totalCount- (positiveCount+ negativeCount);

rationMap.getMotivation(emotionSampleMap, motivation);

rationMap.getTrending(emotionSampleMap, trending);

rationMap.getPrediction(emotionSampleMap,prediction);

rationMap.getEmotionRatio(emotionSampleMap, positiveCount, negativeCount, medCount);

rationMap.getMotivationRatio(emotionSampleMap, totalCount);

rationMap.getCorrelationRatio(emotionSampleMap, totalCount);

double emotionRatio = Math.abs(positiveCount/ negativeCount - negativeCount/ positiveCount);

rationMap.getContinusRatio(emotionSampleMap, emotionRatio);

rationMap.getTrendsRatio(emotionSampleMap);

rationMap.getPredictionRatio(emotionSampleMap);

rationMap.getGuessRatio(emotionSampleMap);

rationMap.getSensingRatio(emotionSampleMap);

String[][] DNNMatrix = new String[emotionSampleMap.size()][15];

Iterator<String> Iterator = emotionSampleMap.keySet().iterator();

int count = StablePOS.INT\_ZERO;

while(Iterator.hasNext()) {

String word = Iterator.next();

EmotionSample emotionSample = emotionSampleMap.get(word);

DNNMatrix[count][StablePOS.INT\_ZERO]=word;

DNNMatrix[count][StablePOS.INT\_ONE]=emotionSample.getMotivation();

DNNMatrix[count][StablePOS.INT\_TWO]=emotionSample.getTrending();

DNNMatrix[count][StablePOS.INT\_THREE]= emotionSample.getPrediction();

DNNMatrix[count][StablePOS.INT\_FOUR]= StablePOS.EMPTY\_STRING + emotionSample.getPositiveCount();

DNNMatrix[count][StablePOS.INT\_FIVE]= StablePOS.EMPTY\_STRING + emotionSample.getMedCount();

DNNMatrix[count][StablePOS.INT\_SIX]= StablePOS.EMPTY\_STRING + emotionSample.getNegativeCount();

DNNMatrix[count][StablePOS.INT\_SEVEN]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getEmotionRatio()\*10000);

DNNMatrix[count][StablePOS.INT\_EIGHT]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getMotivationRatio()\*100000);

DNNMatrix[count][StablePOS.INT\_NINE]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getCorrelationRatio()\*10000);

DNNMatrix[count][StablePOS.INT\_TEN]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getContinusRatio()\*10);

DNNMatrix[count][StablePOS.INT\_ELEVEN]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getTrendsRatio()\*100000);

DNNMatrix[count][StablePOS.INT\_TWELVE]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getPredictionRatio()\*10000\*1000);

DNNMatrix[count][StablePOS.INT\_THIRTEEN]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getGuessRatio()\*100000\*100000);

DNNMatrix[count][StablePOS.INT\_FOURTEEN]= StablePOS.EMPTY\_STRING + (int)(emotionSample.getSensingRatio()\*100000);

count++;

}

return DNNMatrix;

}

public static void main(String[] argv) throws IOException {

SensingTest sensingTest = new SensingTest();

sensingTest.getMatrix();

}

public String[][] getMatrix(String text, A \_A) throws IOException {

pos= \_A.getPosCnToCn();

EmotionMap emotionMap= \_A.getEmotionMap();

Map<String, Object> positive= emotionMap.getPositiveMap();

Map<String, Object> negative= emotionMap.getNegativeMap();

Map<String, Object> motivation= emotionMap.getMotivationMap();

Map<String, Object> trending= emotionMap.getTrendingMap();

Map<String, Object> prediction= emotionMap.getPredictionMap();

sets= \_A.parserMixedString(text);

Map<Integer, WordFrequency> wordFrequencyMap= \_A.getWordFrequencyByReturnSortMap(sets);

RatioMap rationMap= new RatioMap\_E();

Map<String, EmotionSample> emotionSampleMap= rationMap.getEmotionSampleMap(wordFrequencyMap, positive, negative);

double positiveCount= rationMap.findTotalPositiveCount(emotionSampleMap);

double negativeCount= rationMap.findTotalNegativeCount(emotionSampleMap);

double totalCount= rationMap.findTotalKeyCount(emotionSampleMap);

double medCount= totalCount- (positiveCount+ negativeCount);

rationMap.getMotivation(emotionSampleMap, motivation);

rationMap.getTrending(emotionSampleMap, trending);

rationMap.getPrediction(emotionSampleMap,prediction);

rationMap.getEmotionRatio(emotionSampleMap, positiveCount, negativeCount, medCount);

rationMap.getMotivationRatio(emotionSampleMap, totalCount);

rationMap.getCorrelationRatio(emotionSampleMap,totalCount);

double emotionRatio= Math.abs(positiveCount/negativeCount - negativeCount/positiveCount);

rationMap.getContinusRatio(emotionSampleMap, emotionRatio);

rationMap.getTrendsRatio(emotionSampleMap);

rationMap.getPredictionRatio(emotionSampleMap);

rationMap.getGuessRatio(emotionSampleMap);

rationMap.getSensingRatio(emotionSampleMap);

String[][] DNNMatrix= new String[emotionSampleMap.size()][15];

Iterator<String> Iterator= emotionSampleMap.keySet().iterator();

int count= StablePOS.INT\_ZERO;

while(Iterator.hasNext()) {

String word= Iterator.next();

EmotionSample emotionSample= emotionSampleMap.get(word);

DNNMatrix[count][StablePOS.INT\_ZERO]= word;

DNNMatrix[count][StablePOS.INT\_ONE]= emotionSample.getMotivation();

DNNMatrix[count][StablePOS.INT\_TWO]= emotionSample.getTrending();

DNNMatrix[count][StablePOS.INT\_THREE]= emotionSample.getPrediction();

DNNMatrix[count][StablePOS.INT\_FOUR]= StablePOS.EMPTY\_STRING+ emotionSample.getPositiveCount();

DNNMatrix[count][StablePOS.INT\_FIVE]= StablePOS.EMPTY\_STRING+ emotionSample.getMedCount();

DNNMatrix[count][StablePOS.INT\_SIX]= StablePOS.EMPTY\_STRING+ emotionSample.getNegativeCount();

DNNMatrix[count][StablePOS.INT\_SEVEN]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getEmotionRatio()\* 10000);

DNNMatrix[count][StablePOS.INT\_EIGHT]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getMotivationRatio()\* 100000);

DNNMatrix[count][StablePOS.INT\_NINE]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getCorrelationRatio()\* 10000);

DNNMatrix[count][StablePOS.INT\_TEN]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getContinusRatio()\* 10);

DNNMatrix[count][StablePOS.INT\_ELEVEN]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getTrendsRatio()\* 100000);

DNNMatrix[count][StablePOS.INT\_TWELVE]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getPredictionRatio()\* 10000\* 1000);

DNNMatrix[count][StablePOS.INT\_THIRTEEN]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getGuessRatio()\* 10000\* 10000);

DNNMatrix[count][StablePOS.INT\_FOURTEEN]= StablePOS.EMPTY\_STRING+ (int)(emotionSample.getSensingRatio()\* 100000);

count++;

}

return DNNMatrix;

}

}

DETARNN\_IDETEST, 卷积类

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;

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[][] getIDEMatrixExcludeAnalyzer(A \_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[][] getIDEMatrixExcludeAnalyzer(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]= StableCommon.STRING\_EMPTY popFactor;

ideMatrix[ideMatrixCount][2]

= StableCommon.STRING\_EMPTY dovFactor/ dovCount;// 平均distance数

ideMatrix[ideMatrixCount][3]= StableCommon.STRING\_EMPTY 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 德打印函数, 罗瑶光20210420

return ideMatrix;

}

}

DETAANNTEST, 卷积类

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 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, A \_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, A \_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;

}

}

DETADNNTEST, 卷积类

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] + StableCommon.STRING\_SYMBOL\_PER);

System.out.print(ann[i][1] + StableCommon.STRING\_SYMBOL\_PER);

System.out.print(ann[i][2] + StableCommon.STRING\_SYMBOL\_PER);

System.out.print(ann[i][3] + StableCommon.STRING\_SYMBOL\_PER);

System.out.print(dnn[i][0] + StableCommon.STRING\_SYMBOL\_PER);

System.out.print(dnn[i][3] + StableCommon.STRING\_SYMBOL\_PER);

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.getIDEMatrixExcludeAnalyzer(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;

}

}

第二章 Java数据分析算法引擎系统

第一节 研发说明

德塔Java数据分析算法引擎系统说明书V\_1\_0\_2

起源动机

应用特色

使用方法

功能注解

适用范围

注意

感谢

研发需要清单

第二节 研发笔记

API运行原理Flowchat

罗瑶光小高峰过滤快排4代原理

两种比较领先的排序思维对比

维度卷积计算原理

第三节 研发源码

CheckRange, 区间类

Closing, 卷积类

Dilation, 卷积类

Emboss, 卷积类

Erosion, 卷积类

GetMean, 卷积类

Guassian, 卷积类

HitAndMiss, 卷积类

HoughTransform, 卷积类

Laplacian, 卷积类

Mask, 卷积类

Median, 卷积类

Opening, 卷积类

ReadWritePng, 卷积类

RegionGet, 卷积类

Sobel, 卷积类

Strech, 卷积类

Threshold, 卷积类

CnnMeasure, 卷积类

DETA\_ANN\_HMM, 卷积类

DETA\_DNN, 卷积类

BinarySearch, 排序搜索类

BreadthRun, 排序搜索类

BreadthTreeSearch, 排序搜索类

DepthRun, 排序搜索类

DepthTreeSearch, 排序搜索类

LinerSearch, 排序搜索类

PreorderRun, 排序搜索类

PreorderTreeSearch, 排序搜索类

RandomSearch, 排序搜索类

BinarySort, 排序搜索类

BTreeSort, 排序搜索类

Heap\_1D\_Sort, 排序搜索类

InsertionSort, 排序搜索类

Leaf, 排序搜索类

LinkSort, 排序搜索类

LYGSort, 排序搜索类

OrderEvenSort, 排序搜索类

OTreeSort, 排序搜索类

Quick\_5D\_Sort, 排序搜索类

Quick\_6D\_luoyaoguang\_Sort, 排序搜索类

Quick\_Luoyaoguang\_4D, 排序搜索类

SelectionSort, 排序搜索类

TTreeSort, 排序搜索类

LineCodeOperation, 计算类

PixFloat, 像素处理类

Copy, 复制类

DFT, 卷积类

Laplasian, 卷积类

MaxAndMin, 卷积类

Median, 卷积类

PeakStatistic, 统计类

Proportion, 统计类

Quantification, 统计类

Shehold, 卷积类

Tailor, 统计类

DNA元基索引版本略，养疗经音频时序波的处理应用示例

第三章 德塔ETL人工智能可视化数据流分析引擎系统

第一节 研发说明

德塔ETL可视化数据分析引擎系统说明书说明书

起源动机

简介

使用方法

具体重要功能展示

档案管理功能

流操作中相同逻辑节点重用功能

流操作节点配置功能

异常消息面板

适用范围

注意

感谢

研发需要清单

第二节 研发源码

SaveAndUpdateFile, 文件类

SaveAsANewFile, 文件类

OSGI\_chansfer, 接口类

OSGI\_rigester, 接口类

GUIsample, ETL引擎类

ThisCanvas, 画图引擎类

CheckRange, 区间类

DrawArrow, 画图类

DrawFlashSide, 画图类

DrawSinLine, 画图类

DynamicLineUpdater, 动态画图类

LinkList, 链表类

LinkNode, 链表类

UpdateRelatedLine, 画图类

NodeInfo, 链表类

NodeProject, 链表类

MyPanel, 画图类

CacuString, 字符处理类

NodeShow, 节点类

LinkOSGI, 接口类

NodeOSGI, 接口类

ObjectInterface, 插件接口类

ObjectPanel, 插件接口类

ObjectRun, 插件接口类

ObjectView, 插件接口类

DrawArrowHead, 画图类

DrawNeroCellMask31, 画图类

DrawNeroCellMask32, 画图类

UnicornJSplitPane, 组件类

UnicornSplitPaneUI, 组件类

UnicornTreeCellRenderer, 组件类

UnicornTreeUI, 组件类

DeleteFile, 文件类

arffTransferNodeInterface, 插件类

arffTransferNodePanel, 插件类

arffTransferNodeRun, 插件类

arffTransferNodeView, 插件类

arffNode, 节点类

arffLink, 节点类

DrawNeroCellMask33, 画图类

第三节 DNA元基索引版本

1. ThisCanvas去画面闪烁机制
2. 德塔Socket流可编程[数据库语言](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=36694&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)引擎系统
3. 研发说明

德塔Socket流可编程数据库语言引擎系统

起源动机

简介

使用方法

具体重要功能展示

适用范围

注意

感谢

研发需要清单

第二节 研发笔记

DETASocketPLSQLDatabaseFramework7

DETADatabasePLSQL

DETAPLSQLCommands

CommandsDefinition

CommandSamples

RealWorldSamplesByUsingDETAPLSQLDatabase

Acknowledgement

Logbin

第三节 研发源码

ConfigController, 控制类

DBCategoryController, 控制类

DeleteController, 控制类

InsertController, 控制类

SelectController, 控制类

UpdateController, 控制类

VPC, 事务类

BootVPCS, 事务类

RequestFilterController, 控制类

RequestFixController, 控制类

RequestRecordController, 控制类

ResponseController, 控制类

ServerInitController, 控制类

Sleeper, 事物类

SleeperHall, 事务类

ForwardVision, 事务类

RestMapVision, 事务类

VPCSResponse, 事务类

DatabaseLogHall, 日志类

DetaCacheManager, 日志类

DetaDBBufferCacheManager, 数据类

DetaDBUtil, 数据类

DetaUtil, 数据类

CreateTablesImp, 数据类

DeleteRowsImp, 数据类

InsertRowsImp, 数据类

ExecPLSQLImp, 数据类

PLSQLCommandImp, 数据类

ProcessAggregationPLSQL, 数据类

ProcessConditionPLSQL, 数据类

ProcessGetCulumnsPLSQL, 数据类

ProcessRelationPLSQL, 数据类

Cell, 数据类

SelectJoinRowsImp, 数据类

SelectNestRowsImp, 数据类

SelectRowsImp, 数据类

UpdateJoinRowsImp, 数据类

UpdateRowsImp, 数据类

LoginServiceImpl, 数据类

LoginDAOImpl, 数据类

RestControllerPortImpl, 控制类

RestDBConfigImpl，WEB接口类

RestDBDeleteImpl, WEB接口类

RestDBInsertImpl, WEB接口类

RestDBPLSQLImpl, WEB接口类

RestDBSelectImpl, WEB接口类

RestDBUpdateImpl, WEB接口类

RestLoginPortImpl, WEB接口类

TransactionDelegate, 事务类

第四节 VPCSStandard函数提取，养疗经服务器群的继承函数太多，千篇一律略。

ServerForward\_Standard, 标准数据类

ServerInit\_Standard, 标准数据类

ServerRestMap\_Standard, 标准数据类

ServerSleeper\_Standard, 标准数据类

ServerVPC\_Standard, 标准数据类

VPCSRequest, 标准数据类

VPCSResponse, 标准数据类

第五章 德塔数据结构变量快速转换

第一节 研发说明

起源动机

简介

使用方法

适用范围

注意

感谢

研发需要清单

第二节 研发笔记

DetaDataSwapDetailsMapV1.0

德塔数据结构变量快速转换引擎系统复杂点解析1.0

API使用流程FlowChat

混合数列排序FlowChat

第三节 研发源码

ArraySwap, 数组类

ArrayValidation, 数组类

CSVSwap, Office类9

DateSwap, 时间类

DateValidation, 时间类

HashSwap, 哈希类

HttpUnicode, WEB类

ImageSwap, 图片类

IteratorSwap, heap类

JsonSwap, 字码类

ListSwap, 链表类

ListValidation, 链表类

MapSwap, 图类

MatrixSwap, 矩阵类

MatrixValidation, 矩阵类

Matrix3DSwap, 矩阵类

ObjectSwap, 对象类

StockCode, 股市类

QuickLuoyaoguang4D, 排序类

StringSequency, 字符频率类

StringValidation, 字符类

StringSwap, 字符类

StringBuilderSwap, 字符类

TreeSwap, 图类

TSP, 商旅类

TSPEuler, 商旅类

YaoguangEulerTSP, 商旅类

TXTSwap, 文本类

VectorSwap, 向量类

XMLSwap, 脚本类

第四节 DNA元基索引版本

第六章 数据预测引擎系统

第一节 研发说明

起源动机

数据预测引擎系统说明书说明书

简介

使用方法

具体重要功能展示

档案管理功能

动态识别眼睛例子

算法搜索的NLP匹配打分

适用范围

注意

感谢

研发需要清单

第二节 研发笔记

关于核心算法欧拉森林商旅路径思想解析

关于核心算法坐标团重心轨迹算法思想图解

关于核心算法压强斥力和运动轨迹思想

关于核心算法切裂算法思想导图

关于核心算法内部分子相互斥力计算思想

关于核心算法雷达边缘路径计算思想

第三节 研发源码

Fissile, 坐标裂类

FissileWithMatch, 坐标裂类

FuzzProbabailityClassification, 分类类

PositionClasification, 分类类

PositionHeartsSample, 坐标类

ProbabilityClasification, 概率分类类

ClusterAttraction, 簇类

Fusion, 融类

FusionHeart, 融坐标类第517页

SideEnd, 边坐标类

FindHeartPositions, 坐标类

FindMidPositions, 坐标类

ErrorAsserts, 检测类

PositionsHintDirection, 隐坐标类

CorrelationICA, 分析类

ImagePixClassification, 像素类

ImagePixExtract, 像素类

ImagePixGroupFilter, 像素类

ForestIsolation, 坐标观察类

Isolation, 观察类

MatrixIsolationFilter, 观察类

IsIssueDate, 检测类

NLPTopicMatch, 语言处理类

FindPCAMeanDistance, 分析类

FindPositionsGroupPascalHearts, 坐标趋势类

FusionPCAFilter, 过滤类

PCAMeanOfFuzzPC, 分析类

PCAPositionFilter, 过滤类

DistanceRatio, 比率类

LYG4DWithDoubleQuickSort4D, 排序类

TraceFissilePositionHearts, 轨迹类

TracePositionHearts, 轨迹类

YaoguangLuoEulerRingTSP, 商旅类

YaoguangLuoEulerRingTSP2D, 商旅类

第四节 DNA元基索引版本

TraceFissile\_AMV\_MVS\_VSQ\_ByHearts, 商旅类

第七章 类人DNA与神经元基于催化算子映射编码方式

1 DETA humanoid cognition

1.1DETAhumanoidcognitionhistory, 德塔类人认知历史

1.2DETAhumanoidcognitiondevelopment, 德塔类人认知研发

1.3 DETA humanoid cognition application, 德塔类人认知应用

2 DETA Business back end logic

2.1DETABusinessbackendlogichistory, 德塔商业后端逻辑历史

2.2DETABusinessbackendlogicdevelopment, 德塔商业后端逻辑发展

2.3DETABusinessbackendlogicapplication, 德塔商业后端逻辑应用

3 DETA Catalytic computing

3.1DETACatalyticcomputinghistory, 德塔催化计算历史

3.2DETACatalyticcomputingdevelopment, 德塔催化计算发展

3.3DETACatalyticcomputingapplication, 德塔催化计算应用

4 DETA Finding initions

4.1DETAFindinginitionshistory, 德塔催化计算算子单元寻找历史

4.2DETAFindinginitionsdevelopment, 德塔催化计算算子单元寻找发展

4.3DETAFindinginitionsapplication, 德塔催化计算算子单元寻找应用

5 DETA DNA decoding

5.1DETADNAdecodinghistory, 德塔催化单元的DNA解码历史

5.2DETADNAdecodingdevelopment, 德塔催化单元的DNA解码发展

5.3DETADNAdecodingapplication, 德塔催化单元的DNA解码应用

6 IDUC DNA and Its Applications, IDUC DNA与它的应用

7 IDUC VPCS AOPM 3D Nero Cell and Its Applications, 3维神经建模与应用

8 Refer

第八章 肽展公式推导与元基编码进化计算以及它的应用发现

1 DETA INITONS classify/德塔元基分类

2 DETA INITONS PDN words root/德塔元基分类词根

3 DETA INITONS PDN words/德塔元基分类词典

4 DETA TVM/德塔词典肽翻译虚拟机

5 DETA TVM applications/德塔肽翻译虚拟机应用技术

6 DETA TVM PDC/虚拟机应用优化

7 DETA TVM PDE/德塔肽翻译推导

8 DETA TVM PDC functions/德塔肽推导函数化

9 DETA TVM PDC function optimization and PDE/德塔肽推导函数逻辑优化

10 DETA TVM PDE Logic/德塔肽推导函数逻辑优化成肽展公式化

11 DETA TVM PDE and its application/德塔肽展公式应用论证技术

12 TVM humanoid life Research/应用在类人生命进化中

13 Eternal Research/应用在类人生命永生探索领域

14 Not the End/似乎刚刚开始…

15 Conclusion

16 Reference

17 Thanks

第九章 DNA催化与肽展计算和AOPM-TXH-VECS-IDUQ元基解码

1.推导与定义:甲基胞嘧啶在DNA编码和肽计算中具体定义为IDUQ-U变嘧啶

2.推导与定义:2氨基腺嘌呤在DNA编码和肽计算中具体定义为VECS-V变感腺嘌呤

3.推导与定义:次黄嘌呤在DNA编码和肽计算中具体定义为VECS-E尿变嘌呤

4.推导与定义:AOPM-A变胸腺苷, AOPM-O尿胞变腺苷, AOPM-P尿胞变鸟苷, AOPM-M鸟腺苷的S形螺旋 纹血氧峰触发器分子式催化计算严谨完整过程

5.推导与定义:VECS-VECS嘌呤对, VECS嘌呤弧, VECS-IDUQ碱基对, IDUQ-IDUQ嘧啶对的催化模型

6.推导与定义:次黄嘌呤, 尿变嘌呤VECS-E=IDUQ-U变嘧啶, 甲基胞嘧啶E=U全新DNA计算碱基对

7.推导与定义:2氨基腺嘌呤, 变感腺嘌呤VECS-V=IDUQ-I尿嘧啶V-I计算碱基对

8.推导与定义:碱基对Rotation观测与黄嘌呤在DNA编码和肽计算中具体定义为VECS-EC尿变鸟嘌呤

9.推导与定义:尿变鸟嘌呤, 黄嘌呤肽展计算AOPM-OP-T变感腺尿变苷与AOPM-OP-X变感腺鸟苷

10.归纳与定义:DNA与TX-H-U元基解码

11.推导与定义:DNA元基催化计算与ETL肽展神经网络计算流

12.似乎又没有结束, 后序与感谢

13.参考Refer

第十章 DNA非卷积视觉技术

第一节 DNA视觉的动机

第二节 DNA视觉的应用需求

第三节 DNA视觉的具体描述

第四节 RangePDI, 第1074~页

第五节 DNA视觉的应用实现

骨科X片分层DNA边缘填充元基计算应用

DNA肽特征混合蓝光过滤

DNA卷积的动机

DNA卷积的应用需求

第六节 DNA卷积的具体描述

MONITORXCDX元基新陈代谢

Monitor\_XCDX\_Animation, 动画类

Monitor\_XCDX\_Animation\_Pde, 动画类

Monitor\_XCDX\_Animation\_PcfButton, 动画类

Monitor\_XCDX\_Animation\_Pca, 动画类

Monitor\_XCDX\_Animation\_Ica, 动画类

Monitor\_XCDX\_Animation\_EyeScan, 动画类13

第七节 DNA卷积的应用实现

元基魔方

元基神经网络DNN卷ETL流脑计算模型

DNAETL第二代计算模型

费洛蒙 与 元基编码解码方式思维探索

第十一章 DNAETL与元基索引ETL中文脚本编译机

第一节 DNAETL的动机

第二节 DNAETL的应用需求

第三节 DNAETL的具体描述

dNA3DShowNodeASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI, 接口类

OSU\_AVQ\_ASQ\_ASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI, 接口类

第四节 DNAETL的应用实现

JAVA文件肽化

元基索引ETL中文脚本编译机源码与图解

第五节 软件介绍

软件开发动机

软件开发目的

软件价值

软件主要功能

软件开发系统环境

硬件开发系统环境

软件开发软件环境

软件开发硬件环境

软件部署软件环境

软件部署硬件环境

软件办公环境

软件使用方法

软件执行逻辑

软件注意细节

软件申明

软件大小

软件的设计思维

软件的架构理念

第六节 软件源码

E\_PL\_XA\_E

P\_AO\_PLETL

P\_AO\_PL\_XA

P\_AO\_PLTCP

P\_CO\_PL\_XA\_XCDX\_Cache

P\_CO\_PL\_XA\_XCDX\_Kernel

P\_CO\_PL\_XA\_XCDX\_Map

P\_CO\_PL\_XA\_XCDX

P\_I\_CulumnsPL\_XA

P\_RelationPL\_XA

PL\_XA\_Command\_E

SortStringDemo

PL\_XA\_C

PL\_XA\_E

XA\_ShellQ\_JoinRows\_E

XA\_ShellTable

XA\_ShellTables

ShellJPanel

OSGI\_chansfer

OSI\_OSU\_ASQ\_OCQ\_OSI\_PCI\_PCU\_MCI\_MCU\_MSI\_register

I\_TinShellRun

TinMap

App\_CM

LYG10DWCMSSort15D\_XCDX\_C\_U\_A

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C

第十二章 DNA语料数据库加密技术

第一节 DNA语料库的动机

PDEInitonsTVM人类词汇元基词根

生化词根模式

双元组合索引, 元基对生化组合词根模式

第二节 DNA语料库的应用需求

第三节 DNA语料库的具体描述

第四节 DNA语料库的应用实现

语义词汇模式

第五节 DNA加密的动机

第六节 DNA加密的应用需求

FullDNATokenPDI, 肽展类

PDE\_Decrement\_FullDNAFormular, 肽展类

PDE\_Increment\_FullDNAFormular, 肽展类

1. DNA加密的具体描述

第八节 DNA加密的应用实现

TokenUtil, 令牌钥类

TokenPDI, 令牌钥类

PDE\_Decrement\_Formular, 肽展类

PDE\_Increment\_Formular, 肽展类

1. DNA数据库的动机

第十节 DNA数据库的应用需求

第十一节 DNA数据库的具体描述

PDE\_Formular, 智慧数据库脚本ORM语言

PLETLImpl, 语言类

PLETLIntef, 语言类14

PLORMImpl, 语言类

Const, 语言类

Create, 语言类

第十二节 DNA数据库的应用实现

DNA数据库函数分类

DNA数据库特征隐写

DNA数据库文件安全物理加密

DNA数据库数据加密

Token元基概率钥匙归纳

第十三节 智慧数据库语言脚本元基新陈代谢

DictionaryPLSQLStandard, 数据语言类

PLSQLEngineXCDX

E\_PLSQL\_E, 数据语言类

P\_ConditionPLSQL\_XCDX\_Table, 数据语言类

P\_ConditionPLSQL\_XCDX\_Map, 数据语言类

P\_ConditionPLSQL\_XCDX\_Kernel, 数据语言类

P\_ConditionPLSQL\_XCDX\_Cache, 数据语言类

DetaDNAIndex&PLSQLORM增删改查Demo

第十三章 DNA数术推导与RNA\_X\_THF\_DD元基芯片与肽逻辑

第一节 DNA数术的动机

第二节 DNA数术的应用需求

第三节 DNA数术的具体描述

元基数术, 活性, 腐蚀性排序表

元基语义五行排序图

元基语义排序罗盘

元基语义肽展活性排序罗盘

元基肽展公式关系图

元基腐蚀性排序罗盘

语义生化双元基叠加罗盘

无机罗盘术数

生化钥匙罗盘

语义钥匙罗盘

第四节 DNA数术的应用实现

DecadeToPDS, 进制类

PDE\_PDS\_DL, 肽展类

1. 全嘌呤的推导

DCPE THOS MAXF VIUQ 十六进制推导

FU 全嘌呤变嘧啶数字锁存逻辑

十六元基进制的数字逻辑与离散数学发散

第十四章 DNA搜索与筛选应用

第一节 DNA搜索的动机

第二节 DNA搜索的应用需求

第三节 DNA搜索的具体描述

ZhongYaoSearch, 搜索类15

DNA搜索的应用实现

第四节 DNA筛选的动机

第五节 DNA筛选的应用需求

第六节 DNA筛选的具体描述

味觉语义元基定义

味觉生化元基定义

双元筛选索引词库

第七节DNA筛选的应用实现

第八节DNN分词 词汇花函数源码

方剂森林花JOGL三维计算展示函数

药材功效花JOGL三维计算展示函数

药材禁忌花JOGL三维计算展示函数

花的筛选与观测

第十五章 元基模拟染色体新陈代谢催化编码

第一节 元基造字

Q\_OulerRing, 欧拉路径类

LYG9DWithDoubleTopSort4D, 极速排序算法

LYG9DWithDoubleTopSort4D\_U, 极速排序算法

Top Sort 5D

第二节 最新笔记 包含十六元基造字

二次元基新陈代谢方式

LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

第三节 图片识别

图片读脏能力

SkinPathDetectDis, 肽展图片处理类

GetICAStatisticRatio, 肽展图片处理类

GetColorRatioScore, 肽展图片处理类

SkinPathDetectTrip, 肽展图片处理类

MakeImag, 肽展图片处理类

第四节 元基枝与元基花及其在分词，排序，索引，加密上的应用

LYG9DWithDoubleTopSort4D，极速象契混合排序

LYG10DWCMSSort15D\_XCDX\_P\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_U\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_S, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_P\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_S\_C, 象契字符排序类

LYG10DWCMSSort15D\_XCDX\_C\_U\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_U\_A\_C, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A, 象契字符排序类

LYG10DWCMSSort13D\_XCDX\_C\_A\_C, 象契字符排序类

AE\_XCDX\_Map, 肽展中文分词类

AE, 肽展中文分词类

A, 肽展中文分词类

A\_XCDX\_Map, 肽展中文分词类

BinaryForest, 肽展分词索引类

BinaryForest\_A, 肽展分词索引类

CogsBinaryForest\_AE, 肽展分词索引类

CogsBinaryForest\_A, 肽展分词索引类

BinaryForest\_AE, 肽展分词索引类

Nlp\_CE\_XCDX\_A, 肽展分词索引类

Nlp\_C\_XCDX\_A, 肽展分词索引类

Nlp\_CE\_XCDX\_S, 肽展分词索引类

Nlp\_CE\_XCDX\_A, 肽展分词索引类

Nlp\_C\_XCDX\_S, 肽展分词索引类

Nlp\_CE\_XCDX, 肽展分词索引类

POS\_C\_Cognition\_E, 肽展分词索引类

POS\_C, 肽展分词索引类

Pos\_CE\_XCDX\_E, 肽展分词索引类

Pos\_CE\_XCDX\_O, 肽展分词索引类

Pos\_C\_XCDX\_E, 肽展分词索引类

Pos\_C\_XCDX\_O, 肽展分词索引类

Pos\_CE\_XCDX\_P, 肽展分词索引类

Pos\_CE\_XCDX\_E, 肽展分词索引类

Pos\_C\_XCDX\_P, 肽展分词索引类

1. Pos\_CE\_XCDX, 肽展分词索引类
2. TinShell插件\_元基花模拟染色体组计算索引系统

第一节 软件介绍

软件开发动机

软件开发目的

软件价值

软件主要功能

软件开发系统环境

硬件开发系统环境

软件开发软件环境

软件开发硬件环境

软件部署软件环境

软件部署硬件环境

软件办公环境

软件使用方法

软件执行逻辑

软件注意细节

软件申明

软件大小

软件的设计思维

软件的架构理念

第二节 软件源码

RangePDI

LYG9DWithDoubleTopSort5D

StaticRootMap

StaticClassMap

StaticFunctionMap

StaticFunctionMapA\_VECS\_C

StaticFunctionMapA\_IDUQ\_C

StaticFunctionMapO\_VECS\_C

StaticFunctionMapO\_IDUQ\_C

StaticFunctionMapP\_VECS\_C

StaticFunctionMapP\_IDUQ\_C

StaticFunctionMapM\_VECS\_C

StaticFunctionMapM\_IDUQ\_C

StaticFunctionMapA\_VECS\_E

StaticFunctionMapA\_IDUQ\_E

StaticFunctionMapO\_VECS\_E

StaticFunctionMapO\_IDUQ\_E

StaticFunctionMapP\_VECS\_E

StaticFunctionMapP\_IDUQ\_E

StaticFunctionMapM\_VECS\_E

StaticFunctionMapM\_IDUQ\_E

StaticFunctionMapV\_AOPM\_C

StaticFunctionMapV\_IDUQ\_C

StaticFunctionMapE\_AOPM\_C

StaticFunctionMapE\_IDUQ\_C

StaticFunctionMapC\_AOPM\_C

StaticFunctionMapC\_IDUQ\_C

StaticFunctionMapS\_AOPM\_C

StaticFunctionMapS\_IDUQ\_C

StaticFunctionMapV\_AOPM\_E

StaticFunctionMapV\_IDUQ\_E

StaticFunctionMapE\_AOPM\_E

StaticFunctionMapE\_IDUQ\_E

StaticFunctionMapC\_AOPM\_E

StaticFunctionMapC\_IDUQ\_E

StaticFunctionMapS\_AOPM\_E

StaticFunctionMapS\_IDUQ\_E

StaticFunctionMapI\_VECS\_C

StaticFunctionMapI\_AOPM\_C

StaticFunctionMapD\_VECS\_C

StaticFunctionMapD\_AOPM\_C

StaticFunctionMapU\_VECS\_C

StaticFunctionMapU\_AOPM\_C

StaticFunctionMapQ\_VECS\_C

StaticFunctionMapQ\_AOPM\_C

StaticFunctionMapI\_VECS\_E

StaticFunctionMapI\_AOPM\_E

StaticFunctionMapD\_VECS\_E

StaticFunctionMapD\_AOPM\_E

StaticFunctionMapU\_VECS\_E

StaticFunctionMapU\_AOPM\_E

StaticFunctionMapQ\_VECS\_E

StaticFunctionMapQ\_AOPM\_E

序列化索引调用真实示例

第十七章 后序DEMOS

登陆token

肽展session注册

登陆状态验证

PDESwapTestDemo

极速象契拼音笔画排序

精度中文搜索示例

人眼识别的方式

VPCS服务器部署

数字生命

引用

DNA元基催化与肽计算编码

DNA元基催化与肽计算肽展

DNA元基催化与肽计算解码

DNA元基催化与肽计算养疗经应用研究

德塔华瑞集养疗经软件工程类源码引用综合表