ООО «ЛАН-ПРОЕКТ»

Руководство по эксплуатации «ЛАН.Текстовый процессор»

2021

1 Общие сведения

Программа «ЛАН.Текстовый процессор» разработана для извлечения, обработки и анализа текстовых данных из различных источников и на различных языках.

«ЛАН.Текстовый процессор» является программной библиотекой и не имеет графического интерфейса. Далее приводится описание библиотеки.

2 Описание библиотеки «LingLibrary.Net»

2.1 Интерфейсы

IClsDocMetaInfo

Интерфейс для получения метаинформации в кэше классификаторов.

Поля:

IClsDocMetaInfo.MetaInfoId – Получает Id метаинформации в кэше классификаторов.

IGetWordsForm

Интерфейс для трансформации и получения форм слова

Методы:

IGetWordsForm.Metaphone(string word) – Преобразование слова по правилам метафона

- word – слово.

IGetWordsForm.GetAllWordForms(string word, bool ex) – Получение всех форм слова

- word – слово;
- ex – значение, показывающее, использовать ли алгоритм предсказания формы.

IGetWordsForm.GetWordForm(string word, WordGrammarType wordType, WordGrammarCase resultCase, WordGrammarNumber resultNumber, WordGrammarCase sourceCase) – Получение заданной формы слова.

- word – Исходное слово;
- wordType – Тип исходного слова;
- resultCase – Падеж, в который необходимо установить слово;
- resultNumber – Число, в которое необходимо поставить слово;
- sourceCase – Исходный падеж слова.

IGetWordsForm.GetWordFormFromNorm(string word, WordGrammarType wordType, WordGrammarCase resultCase, WordGrammarNumber resultNumber, WordGrammarCase sourceCase) – Получение заданной формы слова из нормальной формы слова.

- word – Исходное слово в нормальной форме;
- wordType – Тип исходного слова;
- resultCase – Падеж, в который необходимо установить слово;
- resultNumber – Число, в которое необходимо поставить слово;
- sourceCase – Исходный падеж слова.

IGetWordsForm.GetPhraseForm(string phrase, WordGrammarCase resultCase, WordGrammarNumber resultNumber) – Получение формы фразы.

- phrase – Исходная фраза в именительном падеже;
- resultCase – Падеж, в который необходимо установить фразу;
- resultNumber – Число, в которое необходимо поставить фразу.

IGetWordsForm.GetAllForms(FullNameGrammarCase sourceName) – Получение всех падежей полного имени.

- sourceName – Исходное полное имя.

IGetWordsForm.GetAllFormsFromNorm(FullNameGrammarCase sourceName) – Получение всех падежей полного имени из нормальной исходной формы.

- sourceName – Исходное полное имя.

IGetWordsForm.GetAllForms(PhraseGrammarCase sourcePhrase) – Получение всех падежей фразы.

- sourcePhrase – Исходная фраза.

IGetWordsForm.GetFullNameForm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы полного имени.

- sourceName – Исходное полное имя;
- grammarCase – Падеж, в который необходимо установить имя;
- grammarNumber – Число, в которое необходимо поставить имя.

IGetWordsForm.GetFullNameFormFromNorm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы полного имени из нормальной формы.

- sourceName – Исходное полное имя;
- grammarCase – Падеж, в который необходимо установить имя;
- grammarNumber – Число, в которое необходимо поставить имя.

IGetWordsForm.GetFullNameForm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase) – Получение формы полного имени в единственном числе.

- sourceName – Исходное полное имя;
- grammarCase – Падеж, в который необходимо установить имя.

IGetWordsForm.GetFullNameFormFromNorm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase) – Получение формы полного имени в единственном числе из нормальной формы.

- sourceName – Исходное полное имя;
- grammarCase – Падеж, в который необходимо установить имя.

IGetWordsForm.GetPhraseForm(PhraseGrammarCase phrase, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы фразы.

- phrase – Исходная фраза;
- grammarCase – Падеж, в который необходимо установить фразу;
- grammarNumber – Число, в которую необходимо установить фразу.

IGetWordsForm.GetPhraseForm(PhraseGrammarCase phrase, WordGrammarCase grammarCase) – Получение формы фразы в единственном числе.

- phrase – Исходная фраза;
- grammarCase – Падеж, в который необходимо установить фразу.

IQueryAnalyzer

Интерфейс проверки запроса на ошибки.

Методы:

IQueryAnalyzer.Check(string query) – Проверить запрос

- query – Запрос на проверку.

IQueryAnalyzer.FullCheck(string localQuery, string globalQuery, System.Threading.CancellationToken token) – проверить запрос на наличие грамматических ошибок и зацикливания

- localQuery – Локальный запрос на проверку;
- globalQuery – Глобальный запрос на проверку;

- token – Механизм прерывания.

IQueryAnalyzer.GetErrorMessage() – Вернуть описание ошибок в виде строки

IQueryAnalyzer.GetMessagesList() – Вернуть описание ошибок в виде списка

IQueryAnalyzer.InitializeProgressCallBack(IProgressCallback callBack) – Привязать к процессу механизм обратного вызова

- callBack – реализация механизма обратного вызова.

IQueryExecuteOnText

Интерфейс для класса, производящего разбор запроса, и выполняющего разметку текста по запросу, выделение атрибутов и объектов из текста

Методы:

IQueryExecuteOnText.ReInit(string localQuery, string globalQuery, IProgressCallback progress) – Инициализировать запрос

- localQuery – локальный запрос;
- globalQuery – глобальный запрос;
- progress – механизм обратного вызова.

IQueryExecuteOnText.MarkupText(string projectPath, string shortFileName, string text) – Получить размеченный текст

- projectPath – путь к папке проекта;
- shortFileName – имя файла;
- text – текст.

IQueryExecuteOnText.GetSemanticDescription(string projectPath, string shortFileName, string text) – Получить описание объектов и атрибутов текста по запросу

- projectPath – путь к папке проекта;
- shortFileName – имя файла;
- text – текст.

IClassificator

Интерфейс классификатора.

Поля:

IClassificator.Name – Получает название классификатора

IClassificator.Guid – Получает GUID Классификатора

IClassificator.Rubrics – Получает словарь рубрик, где ключ – Guid рубрики, значение – название рубрики

Методы:

IClassificator.Classify(string text) – Классифицировать текст

- text – Текст для классификации.

IClassificator.Classify(string text, string metainfo) – Классифицировать текст Функция работет быстрее предыдущей, так как внутри не строится метаинформация. ВАЖНО! Текст должен соответствовать метоинформации. Внутри проверки на соответствие не делается

- text – Текст для классификации;
- metainfo – Метаинформация по тексту.

IClassificator.Classify(string text, IClsDocMetaInfo metaInfo) – Классифицировать текст Функция работет быстрее предыдущей, так как внутри не строится метаинформация. ВАЖНО! Текст должен соответствовать метоинформации. Внутри проверки на соответствие не делается

- text – Текст для классификации;
- metaInfo – Метаинформация по тексту.

IClassificator.ClassifyPack(string text, IClsDocMetaInfo metainfo, string sectionSeparator) – Классифицировать нескольких текстов, предварительно скомбинированный с разделителями. Используется разделитель вида SECTION_separator.

- text – Скомбинированный текст с разделителями;
- metainfo – Метаинформация для комбинированного текста;
- sectionSeparator – Имя разделителя.

2.2 Перечисления

DataModel.ClusterIdErrorStatus

Статус ID кластера, к которому отнесен документ.

DataModel.ClusterIdErrorStatus.IsOk – ID кластера, к которому отнесен документ, валидный.

DataModel.ClusterIdErrorStatus.IsError – ID кластера, к которому отнесен документ, соответствует неизвестной ошибке.

DataModel.ClusterIdErrorStatus.IsDuplicate – ID кластера, к которому отнесен документ, соответствует документу-дубликату.

DataModel.DateTimeDirection

Указание для определения даты в будущем, настоящем или прошедшем времени.

DataModel.DateTimeDirection.Undefined – Дата на текущий момент.

DataModel.DateTimeDirection.Backward – Дата в прошедшем времени.

DataModel.DateTimeDirection.Forward – Дата в будущем времени.

DataModel.WordGrammarCase

Падеж слова.

DataModel.WordGrammarCase.Nominative – RusNominative – именительный

DataModel.WordGrammarCase.Genitive – RusGenitive – родительный

DataModel.WordGrammarCase.Dative – RusDative – дательный

DataModel.WordGrammarCase.Accusative – RusAccusative – винительный

DataModel.WordGrammarCase.Instrumental – RusInstrumental – ворительный

DataModel.WordGrammarCase.Prepositional – RusPrepositional – предложный

DataModel.WordGrammarCase.Vocative – RusVocative – изъявительный

DataModel.WordGrammarNumber

Число слова.

DataModel.WordGrammarNumber.Plural – Множественное число.

DataModel.WordGrammarNumber.Singular – Единственное число.

DataModel.WordGrammarType

Тип слова.

DataModel.WordGrammarType.RusAny – тип не известен (выбирается первый из списка омонимов).

DataModel.WordGrammarType.RusInitialism – инициалы.

DataModel.WordGrammarType.RusPatronymic – отчество.

DataModel.WordGrammarType.RusToponym – географическое название.

DataModel.WordGrammarType.RusOrganisation – название организации.

DataModel.WordGrammarType.RusQualitative – числительное.

DataModel.WordGrammarType.RusName – имя.

DataModel.WordGrammarType.RusSurName – фамилия.

DataModel.WordGrammarType.RusImpersonal – неличное.

DataModel.WordGrammarType.RusSlang – сленг.

DataModel.WordGrammarType.RusProfession – профессия.

2.3 Классы

AdvancedClusterizator

Класс для создания файлов статистики подборки для последующего обучения.

Методы:

AdvancedClusterizator(string docPath,string cachePath, string configPath, int blockSize) – Конструктор

- docPath – путь к документам
- cachePath – путь к кэшу
- configPath – путь к конфигурационному файлу
- blockSize – размер блока документов

AdvancedClusterizator.CreateJsonTable(string outPath) – Запись в файл JSON–результата

- outPath – путь к файлу для записи результата

AdvancedClusterizator.CreateEtalonFile(string outPath) – Запись в файл эталонного разбиения

- outPath – путь к файлу для записи результата

Classificator

Класс для выполнения классификации документов.

Поля:

Classificator.Name – Получает название классификатора

Classificator.Guid – Получает GUID Классификатора Classificator.Rubrics – Получает словарь рубрик, где ключ - Guid рубрики, значение - название рубрики

Методы:

Classificator.Init(byte[] binaryClsData) – Инициализация классификатора из бинарных данных.

- binaryClsData – Бинарные данные cls-файла.

Classificator.Init(string name, string guid, byte[] binaryClsData) – Инициализация классификатора из бинарных данных, именем и GUID

- name – Имя классификатора.
- guid – Уникальный идентификатор классификатора.
- binaryClsData – Бинарные данные классификатора.

Classificator.Init(string clsFile) – Инициализация классификатора из CLS-файла.

- clsFile – Путь к CLS-файлу.

Classificator.Classify(string text) – Классифицировать текст

- text – Текст для классификации

Classificator.Classify(string text, string metainfo) – Классифицировать текст. Функция работает быстрее предыдущей, так как внутри не строится метаинформация. ВАЖНО! Текст должен соответствовать метаинформации. Внутри проверки на соответствие не делается.

- text – Текст для классификации.
- metainfo – Метаинформация по тексту.

Classificator.Classify(string text, IClsDocMetaInfo metaInfo) – Классифицировать текст Функция работет быстрее предыдущей, так как внутри не строится метаинформация. ВАЖНО! Текст должен

соответствовать метоинформации. Внутри проверки на соответствие не делается

- text – Текст для классификации
- metaInfo – Метаинформация по тексту

Classificator.ClassifyPack(string text, IClsDocMetaInfo metainfo, string sectionSeparator) – Классифицировать нескольких текстов, предварительно скомбинированный с разделителями. Используется разделитель вида SECTION_separator.

- text – Скомбинированный текст с разделителями.
- metainfo – Метаинформация для комбинированного текста.
- sectionSeparator – Имя разделителя.

Classificator.Dispose – Завершить работу классификатора.

ClsDocMetaInfo

Класс для получения метаинформации в кэше классификатора.

Поля:

ClsDocMetaInfo.MetaInfoId – Получает Id метаинформации в кэше классификаторов.

Методы:

ClsDocMetaInfo.BuildClsMetaInfo(string text) – Создать метаинформацию из текста в кэше классификаторов.

- text – Текст, для которого создается метаинформация.

Clusterizator

Класс, проводящий кластеризацию подборки документов

Поля:

Clusterizator.ClusterizatorPtr – Получает или задает указатель на память, содержащую объект, проводящий кластеризацию подборки документов

Методы:

Clusterizator(string cachePath, string configPath, int blockSize) – Создание объекта класса, проводящего кластеризацию подборки документов

- cachePath – путь к кэшу
- configPath – путь к конфигурационному файлу
- blockSize – размер блока документов

Clusterizator(int blockSize) – Создание объекта класса, проводящего кластеризацию подборки документов

- blockSize – размер блока документов

Clusterizator.RunClusterization(List<string> filesList, IProgressCallback callback, CancellationToken token) – Запускает кластеризацию с возвратом списка ID

- filesList – Список путей к файлам
- callback – Механизм обратного вызова для прогресса
- token – Механизм корректного останова

Clusterizator.RunClusterizationMultiThread(List<string> filesList, uint numOfThreads, IProgressCallback callback, System.Threading.CancellationToken token) – Запускает кластеризацию с возвратом списка ID

- filesList – Список путей к файлам
- numOfThreads – Количество потоков, в которых выполнять кластеризацию.
- callback – Механизм обратного вызова для прогресса
- token – Механизм корректного останова

Clusterizator.GetMesuareForDocumentPair(int ,int) – Получение меры близости для пары документов

- i – Порядковый индекс документа 1

- j – Порядковый индекс документа 2

Clusterizator.GetMesuareForDocumentPair(int ,int ,System.Int64) – Получение меры близости для пары документов

- i – Порядковый индекс документа 1
- j – Порядковый индекс документа 2
- type – тип блока

Clusterizator.Dispose – Освобождение ресурсов

DataModel.DateTimeExtracted

Класс извлеченных даты из строки.

Поля:

DataModel.DateTimeExtracted.FirstDate – Получает дату начала события.

DataModel.DateTimeExtracted.LastDate – Получает дату конца события. DataModel.DateTimeExtracted.IsRange – Показывает, является ли извлеченная дата диапазоном дат.

Методы:

DataModel.DateTimeExtracted(DateTime firstDate, DateTime lastDate) – Конструктор класса.

- • firstDate – Дата начала события.
- • lastDate – Дата конца события.

DataModel.DetectedLanguage

Класс результата детектирования языка и набора символов.

Поля:

DataModel.DetectedLanguage.CharacterSet – Получает набор символов текста. DataModel.DetectedLanguage.LanguageCode – Получает код языка текста.

DataModel.FullNameGrammarCase

Класс ФИО человека в заданном падеже.

Поля:

DataModel.FullNameGrammarCase.GrammarCase – Получает или задает падеж ФИО.

DataModel.FullNameGrammarCase.FirstName – Получает или задает Имя.

DataModel.FullNameGrammarCase.MiddleName – Получает или задает Отчество.

DataModel.FullNameGrammarCase.LastName – Получает или задает Фамилию. Методы:

DataModel.FullNameGrammarCase – Конструктор по умолчанию устанавливает падеж в .

DataModel.LingConvertResult

Класс результата работы конвертеров из SCATQL–запроса.

Поля:

DataModel.LingConvertResult.ConvertResult – Получает результат преобразования из SCATQL.

DataModel.LingConvertResult.ErrorInfo – Получает информацию об ошибке во время преобразования.

DataModel.LingConvertResult.IsSimple – Получает значение, показывающее, что SCATQL–запрос является простым.

DataModel.LingConvertResult.IsError – Получает значение, показывающее, что во время преобразования произошла ошибка.

DataModel.PhraseGrammarCase

Класс фразы в заданном падеже.

Поля:

DataModel.PhraseGrammarCase.GrammarCase – Получает или задает падеж фразы.

DataModel.PhraseGrammarCase.Phrase – Получает или задает фразу.

Методы:

DataModel.PhraseGrammarCase – Конструктор по умолчанию устанавливает падеж в .

DateTimeExtractor

Класс извлечения даты и времени из строки текста.

Методы:

DateTimeExtractor.GetIsoDateTime(string sourceDateTime, DateTime publicationDate, DateTimeDirection direction) – Получить дату и время в ISO формате из строки текста с учетом прошедшего или будущего времени. Направление используется в случае, например, строки с текстом "в прошлый четверг" или "в следующий вторник".

- sourceDateTime – Исходная строка текста с датой и временем.
- publicationDate – Дата публикации документа, относительно которой определяется требуемая дата.
- direction – Направление преобразования строки текста в дату.

Utils.SectionBuilder

Класс построения текста, разделенного на секции

Методы:

Utils.SectionBuilder.SetTitle(string title) – Задает содержимое секции SECTION_TITLE.

- title – Содержимое секции SECTION_TITLE.

Utils.SectionBuilder.SetText(string text) – Задает содержимое секции SECTION_TEXT.

- text – Содержимое секции SECTION_TEXT.

Utils.SectionBuilder.SetComment(string comment) – Задает содержимое секции SECTION_COMMENT.

- comment – Содержимое секции SECTION_COMMENT.

Utils.SectionBuilder.SetUrl(string url) – Задает содержимое секции SECTION_URL.

- url – Содержимое секции SECTION_URL.

Utils.SectionBuilder.SetAttribute(string name, string value) – Задает содержимое секции SECTION_ATTRIBUTES.

- name – Имя атрибута секции SECTION_ATTRIBUTES.
- value – Значение атрибута секции SECTION_ATTRIBUTES.

Utils.SectionBuilder.RemoveFirstAttribute(string name, string value) – Удаляет первый атрибут с заданным именем и значением из секции SECTION_ATTRIBUTES.

- name – Имя атрибута секции SECTION_ATTRIBUTES для удаления.
- value – Значение атрибута секции SECTION_ATTRIBUTES для удаления.

Utils.SectionBuilder.ToWrappedText() – Возвращает текст с установленными секциями SECTION_...

Utils.SimpleProgressCallback

Класс простого callback'а для прогресс–бара.

Поля:

Utils.SimpleProgressCallback.ProgressDone – Получает текущее значение прогресса.

Utils.SimpleProgressCallback.Warning – Получает сообщение об ошибке. Методы:

Utils.SimpleProgressCallback() – Создает новый экземпляр класса Utils.SimpleProgressCallback.OnProgress(long processedData, long totalData, string additionalInfo) – Метод обратного вызова, устанавливающий текущий прогресс операции.

- processedData – Количество обработанных данных.
- totalData – Количество данных всего.

- additionalInfo – Дополнительная информация.

Utils.SimpleProgressCallback.OnWarning(string warning) – Метод обратного вызова, устанавливающий сообщение об ошибке.

- warning – Сообщение об ошибке.

Extensions.LongExtensions

Методы расширения для типа long, которым представлены ID кластеров.

Extensions.LongExtensions.GetClusterIdErrorStatus(long id) – Получить статус ошибки ID кластера, к которому отнесен документ.

- id – ID кластера, к которому отнесен документ.

GetWordsForm

Класс для трансформации и получения форм слова

Методы:

GetWordsForm.GetAllWordForms(string word, bool ex) – Получение всех форм слова

- word – слово
- ex – значение, показывающее, использовать ли алгоритм предсказания формы

GetWordsForm.GetWordForm(string word, WordGrammarType wordType, WordGrammarCase resultCase, WordGrammarNumber resultNumber, WordGrammarCase sourceCase) – Получение заданной формы слова.

- word – Исходное слово.
- wordType – Тип исходного слова.
- resultCase – Падеж, в который необходимо установить слово.
- resultNumber – Число, в которое необходимо поставить слово.
- sourceCase – Исходный падеж слова.

GetWordsForm.GetWordFormFromNorm(string word, WordGrammarType wordType, WordGrammarCase resultCase,

WordGrammarNumber resultNumber, WordGrammarCase sourceCase) – Получение заданной формы слова из нормальной формы слова.

- word – Исходное слово в нормальной форме.
- wordType – Тип исходного слова.
- resultCase – Падеж, в который необходимо установить слово.
- resultNumber – Число, в которое необходимо поставить слово.
- sourceCase – Исходный падеж слова.

GetWordsForm.GetPhraseForm(string phrase, WordGrammarCase resultCase, WordGrammarNumber resultNumber) – Получение формы фразы.

- phrase – Исходная фраза в именительном падеже.
- resultCase – Падеж, в который необходимо установить фразу.
- resultNumber – Число, в которое необходимо поставить фразу.

GetWordsForm.GetAllForms(FullNameGrammarCase sourceName) – Получение всех падежей полного имени.

- sourceName – Исходное полное имя.

GetWordsForm.GetAllFormsFromNorm(FullNameGrammarCase) – Получение всех падежей полного имени из нормальной исходной формы.

- sourceName – Исходное полное имя.

GetWordsForm.GetAllForms(PhraseGrammarCase) – Получение всех падежей фразы.

- sourcePhrase – Исходная фраза.

GetWordsForm.GetFullNameForm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы полного имени.

- sourceName – Исходное полное имя.
- grammarCase – Падеж, в который необходимо установить имя.
- grammarNumber – Число, в которое необходимо поставить имя.

GetWordsForm.GetFullNameFormFromNorm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы полного имени из нормальной формы.

- sourceName – Исходное полное имя.
- grammarCase – Падеж, в который необходимо установить имя.
- grammarNumber – Число, в которое необходимо поставить имя.

GetWordsForm.GetFullNameForm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase) – Получение формы полного имени в единственном числе.

- sourceName – Исходное полное имя.
- grammarCase – Падеж, в который необходимо установить имя.

GetWordsForm.GetFullNameFormFromNorm(FullNameGrammarCase sourceName, WordGrammarCase grammarCase) – Получение формы полного имени в единственном числе из нормальной формы.

- sourceName – Исходное полное имя.
- grammarCase – Падеж, в который необходимо установить имя.

GetWordsForm.GetPhraseForm(PhraseGrammarCase phrase, WordGrammarCase grammarCase, WordGrammarNumber grammarNumber) – Получение формы фразы.

- phrase – Исходная фраза.
- grammarCase – Падеж, в который необходимо установить фразу.
- grammarNumber – Число, в которую необходимо установить фразу.

GetWordsForm.GetPhraseForm(PhraseGrammarCase phrase, WordGrammarCase grammarCase) – Получение формы фразы в единственном числе.

- phrase – Исходная фраза.
- grammarCase – Падеж, в который необходимо установить фразу.

GetWordsForm.Metaphone(string word) – Преобразование слова по правилам метафона

- word – слово

LanguageDetection

Методы по определению языка и набора символов по тексту.

Методы:

LanguageDetection.DetectLanguage(string text) – Определение набора символов и языка для заданного текста.

- text – Текст, для которого необходимо определить язык.

QueryAnalyzer

Проверщик запроса на наличие ошибок.

Методы:

QueryAnalyzer.InitializeProgressCallBack(IProgressCallback callBack) – инициализировать механизм обратного вызова

- callBack – реализация механизма обратного вызова

QueryAnalyzer.Check(string query) – проверить запрос на наличие грамматических ошибок

- query – запрос

QueryAnalyzer.FullCheck(string localQuery, string globalQuery, System.Threading.CancellationToken token) – проверить запрос на наличие грамматических ошибок и зацикливания

- localQuery – локальный запрос
- globalQuery – глобальный запрос
- token – механизм прерывания

QueryAnalyzer.GetErrorMessage() – получить строку с описанием ошибок

QueryAnalyzer.GetMessagesList() – получить список ошибок

QueryFuzzy2SQL

Класс получения xml–структуры слов с возможными опечатками из БД.

Методы:

QueryFuzzy2SQL(string connectionString) – Конструктор класса поиска подбора слов из базы данных по заданным опечаткам

- connectionString – Строка подключения к базе данных

QueryFuzzy2SQL.PreprocessQueryForFuzzyQuery(string xmlQuery, string databaseName, string databaseName) – Заполнение xml строки словами с возможными опечатками

- xmlQuery – Строка xml со словами, заданных оператором с опечатками
- databaseName – Имя базы данных
- tableName – Имя таблицы, в которой подбираются слова

TextModificationMethodsFactory

Методы по модификации и конвертации текстов.

Методы:

TextModificationMethodsFactory.GetDuplicateDetector(string config) – Получение объекта для детектирования дубликатов

- config – конфигурационный файл

TextModificationMethodsFactory.FragmentToRegex(string fragment) – преобразование текста в регулярное выражение

- fragment – текст

TextModificationMethodsFactory.ConvertQuery2SqlQueryC(string query) – Преобразует текст в строку для использования в функциях полнотекстового поиска

- query – Текст

TextModificationMethodsFactory.ConvertQuery2SqlQueryMistakes(string query) – Преобразует текст с ошибочным запросом в строку для использования в функциях полнотекстового поиска

- query – Текст

TextModificationMethodsFactory.GetFuzzyXmlVariants(string query) – Преобразует текст в строку формата "XML" для поиска слов с опечатками

- query – Текст запроса

TextModificationMethodsFactory.ConvertQuery2SQLQueryFuzzy(string query, string wordsVariants) – Преобразует текст с опечатками в строку для использования в функциях полнотекстового поиска

- query – Текст запроса
- wordsVariants – Варианты слов с опечатками в XML

TextModificationMethodsFactory.ConvertQuery2OracleQueryC(string query) – Преобразует текст в строку для использования в функциях полнотекстового поиска

- query – Текст

TextModificationMethodsFactory.ConvertQuery2ElasticQueryC(string query) – Преобразует текст в строку для использования в функциях полнотекстового поиска

- query – Текст

TextModificationMethodsFactory.AttributeHash(string word) – Получить идентификатор от значения атрибута

- word – Значение атрибута

TextModificationMethodsFactory.AttributeHashSimple(string word) – Получить идентификатор от значения атрибута без нормализации и сортировки

- word – Значение атрибута

TextModificationMethodsFactory.AttributeHashCastedToInt32(string word) – Получить идентификатор от значения атрибута

- word – Значение атрибута

TextModificationMethodsFactory.AttributeHashSimpleCastedToInt32(string word) – Получить идентификатор от значения атрибута без нормализации и сортировки

- word – Значение атрибута

TextModificationMethodsFactory.ConvertQuery2RegexQuery3C(string query, string globalDef) – Преобразует поисковой запрос в регулярную строку для поиска

- query – локальный запрос
- globalDef – глобальный запрос

TextModificationMethodsFactory.DocQuerySimilarQuery(string text, int len) – Получение запроса для текста

- text – текст
- len – длина

TextModificationMethodsFactory.GetTerms(string text) – Получение списка терминов из текста

- text – текст

TextModificationMethodsFactory.GetNormForms(string word, bool predict) – Получение нормальных форм заданного слова на русском или английском языке

- word – слово
- predict – использовать ли алгоритм предсказаний

TextModificationMethodsFactory.GetNormText(string sourceText) – Получение текста с преобрахованными к нормальной форме словами.

- sourceText – Исходный текст для преобразования.

TextModificationMethodsFactory.GetNormFormsEx(string word) – Получение нормальных форм заданного слова на русском или английском языке с учетом возможных омонимов для его словоформ

- word – слово

TextModificationMethodsFactory.TextToCollocList(string text) – Конвертирует строку текста в список словосочетаний (словосочетания отделяются друг от друга при помощи '\n'), экранируя недопустимые символы языка запросов

- text – текст

TextModificationMethodsFactory.TextToTermList(string text) – Конвертирует строку текста в список терминов, экранируя недопустимые символы языка запросов

- text – Текст

TextModificationMethodsFactory.DocQueryMarkupAbstract(string text, string query, string def, string tag, int maxLength) – Получение реферата документа с разметкой.

- text – Текст.
- query – Локальный запрос.
- def – Глобальный запрос.
- tag – Класс тега span, который обрамляет размеченный текст.
- maxLength – Длина реферата.

TextModificationMethodsFactory.DocQueryMarkupText(string text, string query, string def, string tag) – Получение разметки текста по запросу.

- text – Текст для разметки.
- query – Запрос, по которому должен быть размечен текст.
- def – Определения переменных (если есть).
- tag – Тег для выделения найденных фрагментов

TextModificationMethodsFactory.DocQueryMarkupMetaInfoFromText(string projectPath, string shortFileName, string localQuery, string globalQuery, string tag, string text, string model) – Получение разметки для документа из проекта классификатора

- projectPath – путь к проекту
- shortFileName – имя файла
- localQuery – локальный запрос
- globalQuery – глобальный запрос
- tag – каким тэгом помечаем найденные фрагменты
- text – входной текст на разметку
- model – модель языка запросов

TextModificationMethodsFactory.IsValidScatqlQuery(string query, string model) – Проверяет, является ли запрос корректным SCATQL–запросом.

- query – Полученный запрос.
- model – Модель языка SCATQL.

TextModificationMethodsFactory.ConvertFragment2ScatQL(string fragment) – Преобразует текстовый фрагмент в запрос ScatQL.

- fragment – Текстовый фрагмент.

LingFactory

Класс, возвращающий интерфейсы доступа

Методы:

LingFactory.CreateClassificator(string clsFile) – Создать классификатор на основе CLS файла

- clsFile – Путь к CLS файлу

LingFactory.CreateClassificator(byte[] clsMetaData) – Создать классификатор на основе бинарных данных классификатора

- clsMetaData – Бинарные данные классификатора

LingFactory.CreateClassificator(string name, string guid, byte[] clsMetaData) – Создать классификатор на основе бинарных данных классификатора

- name – Имя классификатора
- guid – Уникальный идентификатор классификатора
- clsMetaData – Бинарные данные классификатора

LingFactory.CreateClsDocMetaInfo(string text) – Создать метаинформацию из текста в кэше классификаторов.

- text – Текст, для которого создается метаинформация.

LingFactory.CreateMetainfo(string text) – Создать метаинформацию для текста

- text – текст

LingFactory.CreateMetainfo(string text, uint maxSize) – Создать метаинформацию для текста

- text – Текст.
- maxSize – Максимальный размер текста, при котором не вызывается внешнее извлечение метаинформации.

LingFactory.CombineTextsForCls(List<string> texts, string sectionName) – Объединяет несколько текстов в один, чтобы пачкой передать классификатору. Тексты разделяются тегами SECTION_name, где name задается в параметре.

- texts – Список текстов для объединения.
- sectionName – Имя тега SECTION_ для разделения текстов.

LingFactory.ParseMetainfo(string metainfo) – Получить интерфейс метаинформации документа

- metainfo – строка метаинформации документа

LingFactory.CreateFilterRelevanceQuery(string query, string globalQuery) – Получить вычислитель релевантности запроса

- query – локальный запрос
- globalQuery – глобальный запрос

LingFactory.CreateIStatisticPhraseByQuery(string query) – Получить объект, извлекающий фразы из текста по запросу с их контекстом

- query – запрос

LingFactory.CreateQueryAnalyzer – Получить анализатор запроса на предмет ошибок

LingFactory.CreateClusterizator(string docPath, string cachePath, string configPath, int blockSize) – Получить объект, проводящий кластерный анализ документов

- docPath – путь к документам
- cachePath – путь к кэшу
- configPath – путь к файлу конфигурации
- blockSize – размер обрабатываемого блока

LingFactory.CreateClusterizator(string cachePath, string configPath, int blockSize) – Создать объект, проводящий кластерный анализ документов

- cachePath – путь к кэшу
- configPath – путь к файлу конфигурации
- blockSize – размер обрабатываемого блока

LingFactory.CreateGrammaLogger() – Создать объект, возвращающий дерево разбора локального запроса в XML–виде в соответствии с правилами грамматики

LingFactory.GenerateClusterNumber(string fullDocName, long loaDate) – Возвращает сгенерированный номер кластера

- fullDocName – Полный путь к файлу, для которого генерируем номер кластера

- loaDate – Время загрузки или создания файла, если времени загрузки нет (с точностью до 100 мс)

LingFactory.GenerateQueryExecute(string localQuery, string globalQuery, IProgressCallback progress) – Возвращает исполнитель запросов, умеющий получать разметку документа, объекты и атрибуты

- localQuery – локальный запрос
- globalQuery – глобальный запрос
- progress – механизм обратного вызова

LingFactory.CreateGetWordsForm() – Возвращает интерфейс для трансформации и получения форм слова

2.4 Исключения

Exceptions.BadQueryExeption

Исключение, генерируемое в ситуации, когда невозможно разобрать запрос

Методы:

Exceptions.BadQueryExeption(string error) – Конструктор

- error – текст ошибки

Exceptions.ClsBadVersionException

Класс ошибок неправильной версии проекта классификатора.

Методы:

Exceptions.ClsBadVersionException(string clsFile, string clsFile) – Создает экземпляр с сообщением о том, какой файл проекта классификатора вызвал ошибку, и какая версия классификатора в нем используется.

- clsFile – Путь к файлу проекта классификации.
- versionWarinig – Версия файла проекта классификатора.

Exceptions.ClsDocMetaInfoAlreadyBuiltException

Класс исключения, которое возникает, если метаинформация уже построена.

Методы:

Exceptions.ClsDocMetaInfoAlreadyBuiltException() – Создает экземпляр с сообщением о том, что метаинформация для данного экземпляра класса метаинформации уже была построена.

Exceptions.ClsDocMetaInfoNotInitializedException

Класс исключения, которое возникает, если метаинформация не была построена и внесена в кэш.

Методы:

Exceptions.ClsDocMetaInfoNotInitializedException() – Создает экземпляр с сообщением о том, что метаинформация не была построена и внесена в кэш классификаторов.

Exceptions.ClusterizatorConfigNotFoundException

Исключение, генерируемое при отсутствии конфигурационного файла кластеризации при создании экземпляра класса, проводящего кластеризацию подборки документов

Методы:

Exceptions.ClusterizatorConfigNotFoundException(string configPath) – Конструктор

- configPath – путь к отсутствующему конфигурационному файлу

Exceptions.MarkupExeption

Исключение при разметке

Методы:

Exceptions.MarkupExeption(string errorDescription) – Исключение при разметке

- errorDescription – описание проблемы

2.5 Пример программы

```
namespace demo
{
    using System;
    using System.IO;

    using LingLibrary.Net;

    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length < 1)
            {
                Console.Error.WriteLine("Не заданы параметры запуска.");
                PrintUsage();
                return;
            }

            var fileName = args[0];

            if (!File.Exists(fileName))
            {
                Console.Error.WriteLine($"Файл {fileName} не найден");
                return;
            }

            var text = File.ReadAllText(fileName);

            var metaInfo = LingFactory.CreateMetainfo(text);

            if (args.Length == 2)
            {
                File.WriteAllText(args[1], metaInfo);
            }
            else
            {
                Console.WriteLine(metaInfo);
            }
        }

        private static void PrintUsage()
        {
            Console.Error.WriteLine("Запуск:");
            Console.Error.WriteLine("demo.exe input_file [output_file]");
```

```
                    Console.Error.WriteLine("input_file - файл с текстом, для которого должна
быть построена метаинформация. Обязательный параметр.");
                    Console.Error.WriteLine("output_file - файл, в который должна быть сохранена
метаинформация. Необязательный параметр.");
                }
            }
        }
```

3 Описание языка запросов SCATQL

3.1 Общее описание структуры и принципов языка запросов

3.1.1 Принципы

При разработке синтаксиса и операторов языка SCATQL использовались следующие принципы:

1. Принцип декларативного описания правил. Правила классификации описывают результат, а не алгоритм его получения. При таком подходе правила оказываются независимыми от их реализации, что обеспечивает возможность модификации модулей обработки правил без необходимости их изменения, а также применения, как для классификации, так и для поиска информации.

2. Принцип короткой записи часто используемых операций. Синтаксис языка обеспечивает более простую и короткую запись для наиболее часто используемых операций, которые используются для поиска отдельных терминов и простых выражений в текстах. Сложные операции могут описываться с использованием сложных синтаксических конструкций. Реализация данного принципа позволяет использовать язык, как для поиска текстов, так и для создания сложных правил классификации информации.

3. Принцип комплексного описания классификатора. Данный принцип предполагает, что с использованием языка производится полное описание всех компонент классификатора, а также правил классификации и выделения информации.

4. Принцип универсальности. Данный принцип предполагает, что единый язык используется как для выполнения запросов при поиске, так и при классификации и анализе текстов. За счет использования данного

принципа упрощается обучение пользователей, обеспечивается возможность формирования общей базы знаний с правилами обработки информации для решения широкого круга задач по обработке текстов.

5. Использование UNICODE. Для задания правил используется кодировка UNICODE, что позволяет обрабатывать тексты на большинстве современных языков.

3.1.2 Отличительные особенности

Отличительными особенностями языка SCATQL являются:

1. Использование фрагментной модели представления текстов и правил, в рамках которой правила задаются в виде операций над множеством фрагментов текстов. Наличие формального математического описания всех операций и обоснования алгебраических и вычислительных свойств.

2. Поддержка комплексного описания классификаторов, при котором результатом любой операции в нем является множество фрагментов текста, которые описываются координатами начала и конца, весом и признаком истинности.

3. Совместное использование методов классификации, основанных на обучении, на примерах и на задание правил экспертами.

3.1.3 Общая структура операций языка

Все операторы языка можно условно разделить на три уровня. Кроме этого все операторы также можно подразделить на двенадцать основных групп, которые относятся к тому или иному уровню:

1. Уровень основных операторов – осуществляют операции по поиску отдельных элементов текста, выполнению базовых операций над множествами фрагментов, получению их числовых характеристик и строкового представления. К данному уровню относятся следующие группы операторов:

1.1. элементарные операторы;

1.2. базовые операторы (унарные, бинарные и множественные);

1.3. операторы поиска по словарю;

1.4. числовые операторы;

2. Уровень модели предметной области и правил – осуществляют определение понятий и переменных, сложных логических правил, обучаемых и гибридных правил классификации. К данному уровню относятся следующие группы операторов:

2.1. логические операторы;

2.2. операторы определения понятий и словарей;

2.3. операторы задания динамических правил;

2.4. методы машинного обучения.

3. Уровень выходных операторов – осуществляют задание структуры рубрик, атрибутов документов, правил выделения описаний объектов предметной области и порядка формирования выходного результата обработки текста. К данному уровню относятся следующие группы операторов:

3.1. управляющие операторы;

3.2. строковые операторы;

В языке SCATQL предусмотрена возможность написания комментариев.

Комментарии могут быть как однострочными «//» или многострочными «/* */».

Однострочный комментарий - это комментарий, записанный в одну строку и имеющий следующий синтаксис:

```
запрос // текст комментария
```

Пример применения однострочного комментария:

```
$Name //дескриптор, содержащий в себе русскоязычные имена
```

Многострочный комментарий - это комментарий, записанный в несколько строк или ненужный кусок кода (запроса). Он имеет следующий синтаксис:

```
/*
    текст

комментария
*/
```

Пример применения многострочного комментария:

```
$Name :1 $SurName
/*
  дескриптор $Name осуществляет поиск
имен
  дескриптор $SurName осуществляет поиск
фамилий
*/
```

3.2 Описание основных групп операторов языка запроса

3.2.1 Элементарные операторы

Элементарные операторы – операторы, которые находятся на самом низком (начальном) уровне формирования запросов. Операторы данной группы осуществляют поиск и получение фрагментов для отдельных элементов текста (слов, знаков препинания, атрибутов и разделов документа). Они обладают следующими отличительными особенностями:

1. Реализуются с использованием специальных индексов, создаваемых для отдельного документа (метаинформация) или массива документов (полнотекстовые индексы в СУБД, специализированные средства создания полнотекстовых индексов).

2. Служат основой для реализации всех остальных операторов языка и определяют возможности по обработке тех или иных элементов текста.

К этой группе операторов относятся:

norm_term – поиск слова с учетом словоформы;

termplate_word – шаблонная лексема в исходной форме;

source_word – слово в исходной форме с шаблонными символами;

norm_word_ex – словосочетание;

descriptor_term – дескриптор слова;

fuzzy_term – поиск фонетически близких лексем;

lemma_term – лемма слова;

#interval - интервал;

#section - именованный фрагмент документа.

3.2.2 Унарные операторы

1. #maxreduce - выбор фрагментов максимальной длины;
2. #minreduce - выбор фрагментов минимальной длины;
3. #repeat - объединение нескольких соседних выражений в одно;
4. {{ }} - расширенный оператор квантификации.

3.2.3 Бинарные операторы

Бинарные операторы осуществляют формирование нового множества фрагментов путем выполнения операций над двумя исходными множествами фрагментов, полученными с помощью соответствующих подвыражений.

Общая формула записи бинарных операторов имеет следующий вид:

A #operator B,

где A и B множества фрагментов, а #operator – название оператора.

Отдельные операторы могут иметь дополнительные параметры, которые указываются после названия оператора

В данную группу операторов относятся:

#or - операция «ИЛИ»;

#and - операция «И»;

#next – оператор соединения последовательности фрагментов;

#not – проверка отсутствия одного выражения рядом с другим;

#exist – проверка наличия одного выражения рядом с другим;

#agree - проверка согласованности первых слов фрагментов по роду, числу и падежу;

#inter - операция пересечения фрагментов;

#in - оператор проверки включения;

#contains - оператор проверки содержания;

#leftjoin – оператор дополнения фрагментов из первого множества частями фрагментов из второго множества слева;

#rightjoin - дополнение фрагментов из первого множества частями фрагментов из второго множества справа;

3.2.4 Множественные операторы

Множественные операторы осуществляют формирование нового множества фрагментов путем выполнения операций над более чем двумя исходными множествами фрагментов.

Общая формула записи множественных операторов имеет следующий вид:

#operator(A1 A2 … An),

где A1 - An - множество фрагментов, а #operator – название оператора.

К этой группе операторов относятся:

1. #concur - поиск совпадающих выражений;
2. #any - множественная операция «ИЛИ»;
3. #all - множественная операция «И»;
4. #seq - множественная операция последовательности.

3.2.5 Операторы поиска по словарю

Операторы поиска по словарю аналогичны оператору «#any» и увеличивают скорость обработки запроса.

Общая формула записи операторов поиска по словарю такая же, как и для записи множественных операторов, и имеет следующий вид:

#operator(A1 A2 … An),

где A1 - An - множество фрагментов, а #operator – название оператора.

К этой группе операторов относятся:

#normdict - поиск по словарю с нормализацией;

#sourcedict - поиск по словарю в исходной форме.

3.2.6 Числовые операторы

Числовые операторы осуществляют задание весов и получение различных числовых характеристик у множеств фрагментов, кроме этого операторы предназначены для выполнения вычислений над численными значениями в условных операторах.

К этой группе операторов относятся:

#int - задание целочисленной константы (целое число);

#real - задание константы с плавающей точкой (вещественное число);

/, *, +, - - простые арифметические операции (деление/умножение, сложение/вычитание);

#greater - больше (оператор для сравнения значений);

#less - меньше (оператор для сравнения значений);

#notgreater - меньше или равно (оператор для сравнения значений);

#notless - больше или равно (оператор для сравнения значений);

#equal - равно (оператор для сравнения значений);

&& - логическое И

||- логическое ИЛИ

#strlen - длина запроса;

#freq – число элементов в множестве фрагментов (частоты встречаемости);

#max - максимум нескольких выражений;

#min - минимум нескольких выражений;

#sum - сумма нескольких выражений;

#diverse - число подвыражений, которые вернули непустой результат;

#not - признак неистинности выражения (выражение ложно);

#bin - признак истинности выражения (выражение верно);

#nscalar – скалярное произведение вектора весов и вектора частот термов с нормировкой;

#fscalar – скалярное произведение вектора весов и вектора частот термов;

#bscalar – скалярное произведение булева вектора встречаемости термов и вектора их частот.

#wset - задание веса фрагментов;

#wmult - произведение веса фрагментов, соответствующих выражению;

#wsum - суммарный вес фрагментов, соответствующих выражению;

#item - получение фрагмента с заданным номером;

#nterm - число терминов в тексте;

#nsent - число предложений в тексте;

#exp - экспонента;

#lg – десятичный логарифм;

#ln – экспоненциальный логарифм;

#lb – бинарный логарифм;

#sfun - нормирование сигмоидальной функции.

3.2.7 Логические операторы

Логические операторы обеспечивают проверку числовых условий для определения множества выделяемых фрагментов.

К этой группе операторов относятся:

конструкция «#if … #then … #else … #endif» - проверка условий;

конструкция «#ifdef … #else … #endif» - проверка определённости переменной и выполнения соответствующего подзапроса.

3.2.8 Операторы определения понятий и словарей

Операторы определения понятий и словарей предназначены для построения модели предметной области. для задания правил классификации.

К этой группе операторов относятся:

#namespace … #endnamespace - пространства имен;

@ - ссылка на определение;

@@ - ссылка на переменную;

@@@ - ссылка на функцию;

#ref - шаблонная ссылка на переменную;

#define - объявление определения;

#set - объявление переменной;

#function – объявление функции;

#add - дополнение переменной без переопределения;

#use - задание текущего пространства имен;

3.2.9 Операторы задания динамических правил

Операторы предназначены для обеспечения динамического формирования новых правил в зависимости от выделенной в тексте информации другими правилами или в зависимости от заданных пользователем определений понятий.

В данную группу операторов относятся:

#findfch - динамический поиск первой буквы слова;

#findlemma - динамическое склонение слов к нормальной форме;

#findpre - динамический поиск префикса слова;

#findsrc - поиск набора слов в исходной форме;

#concat - объединение наборов строк;

#decode - преобразование текста оператора к строке с преобразованием экранированных символов;

#substr - получение подстроки из текста оператора.

3.2.10 Методы машинного обучения

Алгоритмы машинного обучения предназначены для автоматического формирования правил классификации. Выделяются статистические (вероятностные) методы и методы на основе правил.

Статистические методы основаны на построении многомерного вероятностного распределения на множестве терминов в рубрики.

В методах классификации на основе правил решающая процедура строится в форме логического правила, в котором проверяются условия на вхождение терминов в документ.

К статистическим методам относятся:

SVMTrain – метод опорных векторов;

BernTrain – многомерное распределение Бернулли;

VMFSTrain – распределение фон Мизеса-Фишера;

PolinomTrain – полиномиальное распределение;

Методы на основе правил:

ListTrain – решающий список терминов;

TreeTrain – дерево решений;

MarkupListTrain – решающий список терминов, основанный на пользовательской разметке;

MarkupTreeTrain – дерево решений, основанное на пользовательской разметке;

3.2.11 Строковые операторы

Строковые операторы получают строковое представление множества фрагментов с преобразованием к необходимому виду. Операторы используются для задания атрибутов документа.

Запись строковых операторов имеет следующий вид:

#operator(QUERY),

где #operator – название оператора, «QUERY» - запрос для получения фрагментов текста, которые будут преобразовываться оператором.

В данную группу операторов относятся:

#normstr - строковое представление фрагментов в нормальной форме;

#sourcestr - строковое представление фрагментов в исходной форме (как в тексте);

#strcon - конкатенация строки и списка;

#strmatch - фильтрация списка по полному совпадению;

#strfind - фильтрация списка по частичному совпадению;

#strreg - преобразование строк с помощью регулярных выражений;

#strtrim - удаление пробелов до и после выделяемого фрагмента (тримминг).

3.2.12Управляющие операторы

Управляющие операторы используются для задания отношений между рубриками, управления результатами классификации с учетом

иерархических отношений между рубриками, а также для оценки степени соответствия рассматриваемого текста рубрикам.

В данную группу операторов относятся:

#attribute - определение атрибутов;

#object – определение объектов;

#label – задание именной метки для части правила;

#getlabel – получение фрагментов по метке;

#property – задание свойства объекта;

#relevance - задание пользовательской формулы для вычисления степени релевантности (соответствия) документа правилу;

#replace - задание автоматических подстановок слов.

3.3 Базовые операции и синтаксис SCATQL

Базовые операции – предназначены для поиска и выделения в тексте фрагментов, соответствующих отдельным словам и выражениям. Полное описание синтаксиса операций приведено в таблицах ниже. Рассмотрим только их назначение и примеры использования.

3.3.1 Простые операции

Операция поиска слова во всех словоформах (без оператора) – осуществляется поиск отдельного слова с учетом всех его словоформ. Например, в результате выполнения выражения

```
Россия
```

будут отобраны все тексты, где встречается слово «Россия» вне зависимости от регистра и словоформы, например, «РОССИЯ», «россию», «РосСии», «России» и т.п.

Операция поиска слова в нормальной форме (!) – осуществляется поиск отдельного слова, заданного в нормальной форме, с учетом всех его словоформ. В отличие от простого поиска в данном случае при генерации вариантов склонения слова учитывается то, что оно приведено в нормальной форме. Это позволяет более точно выполнять поиск фамилий. Например, в результате выполнения выражения

```
Иванова
```

будут выделены в тексте словоформы, соответствующие как мужскому («Иванов», «Иванову» и др.), так и женскому роду («Иванова»,

«Ивановой» и др.), а также «Иваново» и т.п. в результате выполнения выражения

```
!Иванова
```

будут выделены только словоформы, соответствующие женскому роду.

Операция поиска слов с похожим звучанием (%) – осуществляется поиск отдельных слов похожих по звучанию на заданное слово. Данный режим особенно необходим для поиска иностранных имен и фамилий. Для этих целей слово сначала преобразуется в нормальную форму, затем выполняется запись слова с использованием букв латинского алфавита и вычисляется так называемый фонетический код с применением метода "Метафон", который является одинаковым для всех слов с одинаковым звучанием. В результате выполнения запроса находятся все слова со звучанием, похожим на заданное слово, как на русском, так и на английском языке. Например, в результате выполнения выражения

```
%Сейдулла
```

будут выделены в тексте такие слова как «Сейдулла», «Сайдулла», «Сайдула» и т.д.

Данный оператор полезно использовать при поиске иностранных имен и фамилий.

Операция поиска слов/словосочетаний в исходной форме ('') – осуществляется поиск и выделение слов только в той форме, в которой они указаны в запросе (словоизменение не производится). Например, в результате выполнения выражения

```
'Иванову'
```

в тексте будут подсвечены только равные указанной словоформе слова. Данный режим целесообразно использовать для поиска аббревиатур, поиска слов в заданном падеже и времени.

Операция поиска шаблонных выражений – осуществляется поиск слов с использованием следующих шаблонных символов:

* – произвольное количество символов;

? – один символ;

[] – множество символов (в скобках можно как перечислить отдельные символы, так и указать интервал символов, путем вставки дефиса между парой символов);

\ – использование специальных символов (для поиска любого нечислового или небуквенного символа перед ним необходимо ставить знак слеш).

Приведем примеры использования шаблонных символов.

Выражение 1

```
'Росси*'
```

находит и выделяет в текстах все слова, которые начинаются на «Росси», например, будут найдены слова «Россия», «российский» и т.п.

Выражение 2

```
'к?ш'
```

находит и выделяет в текстах слова, где второй символ в слове может быть любой.

Выражение 3

```
'к[эе]ш'
```

находит и выделяет в текстах слова «кеш» и «кэш».

Следующие запросы показывают пример поиска специальных символов. Заметим, что если в запросе указывать несколько слов, то их необходимо брать в одинарные кавычки. Например,

```
'100\$'
'ivanov\@mail\.ru'
'01\.02\.2010'
```

Необходимо отметить, что каждый специальный символ рассматривается как отдельное слово. В следующем примере осуществляется поиск и выделение чисел, состоящих из двух цифр.

```
'[0-9][0-9]'
```

Операция поиска отдельных дескрипторов ($) – осуществляется поиск и выделение слов в тексте, которые имеют свойства, соответствующие заданному дескриптору. Например, выражение

```
$FirstUp
```

выделяет все слова в следующем тексте, которые начинаются с большой буквы.

*«**Сегодня** в **Москве** и области синоптики предсказывают дождь и усиленный ветер.»*

Операция поиска слова с набором дескрипторов ([]) – осуществляется поиск слов, которые совпадают с заданным словом и имеют сразу несколько указанных дескрипторов. Например, следующее выражение

```
[$Noun $Locativ]
```

отбирает все существительные в предложном падеже, а запрос

```
[Сегодня $FirstUp]
```

отбирает вхождения слова «Сегодня» во всех формах с большой буквы.

Операция поиска словосочетаний ("") – осуществляется поиск словосочетаний и таких последовательностей слов или выражений, которые следуют непосредственно друг за другом без пропусков. Например, выражение

```
"Российская Федерация"
```

отбирает тексты, в которых содержится последовательность слов «Российская Федерация» во всех возможных формах, т.е. будут выделены такие фрагменты как «Российская Федерация», «Российской Федерации» и др. Необходимо отметить, что при использовании данного оператора можно в качестве его элементов указывать и сложные выражения. Например, выражение

```
(гражданин житель) :1 "Российская Федерация"
```

отбирает тексты, в которых содержатся такие словосочетания как «гражданин Российской Федерации», «житель Российской Федерации», «право на недвижимое имущество» в различных формах.

Операция группировки выражений (()) – данная операция используется для возможности построения сложных операций, которые включают в себя несколько других операций, а также для явного задания последовательности применения операций. Например, выражение

```
(гражданин житель)
```

будет рассматриваться как отдельное слово в других операциях, пример смотрите выше в пункте «Операция поиска словосочетаний».

Операция ИЛИ (пробел) – осуществляется поиск и выделение всех указанных слов или выражений в тексте. Данная операция соответствует логической операции ИЛИ. Например, выражение

```
гражданин житель
```

выделяет в тексте все формы перечисленных в приведенном запросе слов (текст отбирается, если найдено хотя бы одно из слов).

Операция поиска последовательностей слов без контактности (:) – осуществляется поиск и выделение фрагментов, которые содержат слова в заданной последовательности. При этом возможно задание ограничений на максимальное расстояние между словами (выражениями), проверка отсутствия определенных слов перед указанным словом или после него. Необходимо еще раз отметить, что в тексте выделяются фрагменты минимальной длины, которые содержат все указанные слова. Рассмотрим типичные примеры использования данного оператора. Для поиска текстов и выделения в них фрагментов, которые содержат заданные два слова в указанном порядке, можно использовать следующее выражение

```
пожар : Челябинск
```

оно отбирает тексты, в которых после слова «пожар» находится любая форма слова «Челябинск», при этом будут выделяться такие фрагменты как «**пожар** на территории главного завода **Челябинска**», «**пожар** в **Челябинске**» и др.

Операция поиска последовательностей с контактностью (:n) – осуществляется поиск последовательности в заданном порядке с ограничением на максимальное расстояние между терминами. При этом ограничение указывается как в количестве слов, так и в количестве предложений. Например, запрос

```
пожар :3 Челябинск
```

отбирает тексты, в которых слово «Челябинск» стоит максимум третьем по отношению к слову «пожар». Например, фрагмент «**пожар** в **Челябинске**» будет отобран, а фрагмент «**пожар** на территории главного завода **Челябинска**» не. Слово изменение производится.

Для указания расстояния в предложениях необходимо после числа поставить символ «s». Например, запрос

```
пожар :1s Челябинск
```

С помощью специального параметра можно явно указать области действия оператора. В частности, можно задать ограничение, что все слова должны встречаться в одном предложении. Это можно сделать с помощью следующего выражения.

```
пожар :\s Челябинск
```

Операция «И» (&) – осуществляет отбор текстов, которые содержат все указанные слова без учета порядка их следования. Данная операция, так же, как и операция поиска последовательностей слов, имеет ряд параметров, которые позволяют задать ограничения на расстояние между словами или выражениями, проверить наличие или отсутствие определенных выражений рядом с заданным выражением или словом. Рассмотрим типичные варианты его использования. Наиболее простую форму данный оператор имеет в ситуации, когда необходимо проверить наличие в тексте двух заданных выражений, без дополнительных ограничений. Например, выражение

```
пожар & (Челябинск Москва)
```

отбирает тексты, в которых встречается слово «пожар» и одно из слов в скобках. При этом в тексте выделяются фрагменты для каждого из указанных выражений по отдельности.

Операция «И» ограничением на расстояние (&n) – задание ограничения на расстояние между выражениями после символа «&» можно указать расстояние в словах или в предложениях. Например, выражение

```
пожар &5 (Челябинск Москва)
```

отбирает тексты, в которых рядом со словом «пожар» на расстояние не более 5 слов стоит либо слово «Челябинск», либо «Москва» без учета порядка следования, например, «В **Москве** сегодня был **пожар**», «**пожар** в **Челябинске**», «**пожар** на территории **Челябинска**» и т.д.

Для указания расстояния в предложениях необходимо написать, например, следующее выражение

```
пожар &5s (Челябинск Москва)
```

По умолчанию при отсутствии ограничений поиск заданных в операции выражений осуществляется во всем тексте. Для поиска их в одном предложении можно использовать опцию \s.

```
пожар &\s (Челябинск Москва)
```

3.3.2 Операции для поиска сложных выражений

Рассмотрим теперь операции для поиска сложных выражений, которые включают в себя другие операции, связанные различными условиями.

Операция поиска заданной последовательностей с условиями – для поиска текстов, в которых после или перед заданным словом не находится определенное выражение, можно использовать специальный маркер отрицательного условия «^». Данный маркер ставится или справа «: ^» или слева «^ :» от знака «:» в зависимости от того к какому элементу применяется отрицание. Например, запрос

```
пожар :1s^ Челябинск
```

отбирает тексты, в которых после слова «пожар» в том же или следующем предложении не содержится слово «Челябинск».

Запрос

```
пожар ^:1s Челябинск
```

напротив, отбирает тексты, в которых перед словом «Челябинск» в том же или предыдущем предложении не содержится слово «пожар».

В ряде случаев при выделении объектов в тексте необходимо проверить наличие перед или после слова определенных выражений, но при этом не включать их в выделяемый фрагмент. Например, запрос

```
'зам\.директор' ? :1s Иванов
```

отбирает тексты, в которых перед словом «Иванов» в том же или следующем предложении содержится выражение «зам. директор». При этом в тексте будет выделена только фамилия «Иванов».

Операция «И» с условиями – осуществляется проверка наличия или отсутствия определенного выражения осуществляется путем указания после символа «&» символов «?» или символа «^», соответственно. Например, выражение

боевик &\s ^ (издательство книга сюжет издать актер автор режиссер)

отбирает тексты, в которых встречается слово «боевик», но рядом нет слов «издательство книга сюжет издать актер автор режиссер». Такое правило позволяет отделить употребление слова боевик в значении «член незаконного вооруженного формирования» от его употребления в значении названия жанра фильма.

Операция поиска и выделения повторяющихся элементов (+) – осуществляет отбор текстов, в которых заданное выражение повторяется несколько раз подряд. Данная операция допускает задание дополнительного ограничения на расстояние между повторяющимися фрагментами в тексте. Например, выражение

("($Digit '[а-я]') $CloseBracket" : ($PointComma $Point))+1

отбирает тексты, в которых имеются нумерованные перечисления и которые заканчиваются на точку с запятой или на точку. При этом

предполагается, что между элементами перечисления не может быть других слов. В частности, ему соответствует следующий фрагмент:

```
1) организации;
2) физические лица.
```

Выражение

(Алтайский Забайкальский Камчатский Краснодарский Красноярский Пермский Приморский Ставропольский)+2

отбирает тексты, в которых имеются перечисления названий краев, между которыми встречается не более одного слова. В частности, данному правилу соответствует следующий фрагмент Алтайский, Забайкальский и Камчатский, но не соответствует следующее выражение Алтайский край, Забайкальский край и Камчатский край

3.3.3 Операции задания понятий и переменных

Операции задания понятий – предназначены для построения модели предметной области путем задания системы взаимосвязанных понятий и формирования правил классификации на их основе.

Операция определения понятия (#define) – позволяет задать определение (понятие), которое при компиляции (обучении) классификатора будет подставляться вместо ссылки на него. Использование данного оператора позволяет организовать формирование правил классификации путем первоначального создания системы понятий, а затем задания различных логических и контекстуальных условий на встречаемость их в тексте.

Например, выражение

```
#define РАЙОН (район 'р-н' 'р-на' 'р-не' 'р-он' 'р-она' 'роне' 'р-ну')
#define ГОРОД (город г гор)
```

задает понятия «район» и «город» путем перечисления вариантов их написания в тексте.

Операция задания ссылки на определение понятия (@) – данный оператор позволяет ссылаться на ранее определенные понятия. Необходимо отметить, что можно ссылаться как на понятия, определенные в тексте правила для рубрики, так и на понятия, определенные в настройках классификатора (на данные понятия можно ссылаться во всех рубриках). Если одно и тоже понятие определено и в рубрике, и в общих настройках классификатора, то будет использоваться определение, приведенное в рубрике. Рассмотрим примеры использования ссылок на понятия.

Выражение

```
(АМУРСКИЙ &3 @РАЙОН)
(@ГОРОД :1 АМУРСК)
#define РАЙОН (район 'р-н' 'р-на' 'р-не' 'р-он' 'р-она' 'роне' 'р-ну')
#define ГОРОД (город г гор)
```

ссылается на определения двух понятий «ГОРОД» и «РАЙОН», которые далее определяются в том же выражении. В результате компиляции данного выражения на этапе обучения классификатора будет сформировано правило

```
(АМУРСКИЙ &3 (район 'р-н' 'р-на' 'р-не' 'р-он' 'р-она' 'р-оне' 'р-ну'))
((город г гор) :1 АМУРСК)
```

которое отбирает документы про Амурский район. Ссылку на определение понятия можно делать и внутри определения другого понятия.

Операция задания набора фрагментов (#set) – при задании нескольких ссылок на одно определение при компиляции правила его содержимое будет многократно повторено. Это может приводить к существенному возрастанию размера правил и, соответственно, замедлению скорости их выполнения при классификации. Операция задания набора

фрагментов позволяет сохранить множество выделенных фрагментов в отдельной переменной, при обращении к которой вычисления уже не будут делаться повторно. Например, при многократном использовании понятий «РАЙОН» и «ГОРОД» их можно задать в виде множеств фрагментов следующим образом.

```
#set РАЙОН (район 'р-н' 'р-на' 'р-не' 'р-он' 'р-она' 'роне' 'р-ну')
#set ГОРОД (город г гор)
```

Необходимо отметить, что в виде набора фрагментов можно задавать только полностью определенные понятия, абстрактные понятия задавать нельзя, так как множества фрагментов, которые им соответствуют, зависят от контекста их использования. По этой причине абстрактные понятия можно задавать только с помощью оператора #define.

Операция обращения к набору фрагментов (@@) – позволяет обратиться к множеству фрагментов из текста правила. Например, выражение

```
(АМУРСКИЙ &3 @@РАЙОН)
(@@ГОРОД :1 АМУРСК)
#set РАЙОН (район 'р-н' 'р-на' 'р-не' 'р-он' 'р-она' 'роне' 'р-ну')
#set ГОРОД (город г гор)
```

ссылается на понятия «ГОРОД» и «РАЙОН», заданные с помощью наборов фрагментов. Необходимо отметить, что на этапе компиляции данного правила ссылки на наборы фрагментов сохраняются внутри правила. На этапе выполнения при первом обращении к соответствующей ссылке производится нахождение в тексте множества фрагментов, которое ей соответствует. При последующих обращениях к данной ссылке поиск уже не производится, а возвращается сохраненное ранее множество фрагментов.