Rosette 言語・文字コード 判別システム Pure Java版 プログラミングガイド

Version 7.15.0.c58.2

Rosette 言語・文字コード判別システム Pure Java版
プログラミングガイド

発行 May 2016

Basis Technology Corp.

ベイシス･テクノロジー株式会社

One Alewife Center
Cambridge, MA 02140
T 617.386.2000
F 617.386.2020

〒102-0084
東京都千代田区二番町9-6
バウ・エプタ3F
Email support@rosette.com

第1章 はじめに

Rosette 言語・文字コード判別システム Pure Java版は、Rosette 言語・文字コード判別システムを100%Javaで実装し直した製品です。RLI-JEは2種類の判別アルゴリズムを用意しています。

標準解析.　標準解析では入力データ(バッファー、ファイル、文字列)の言語、エンコーディング、文字種を判別します。入力に複数言語を含む場合、言語の領域を認識し、それぞれの言語を判別します。377種類の言語-エンコーディング-文字種の組み合わせに対応しています(55言語、46エンコーディング、18文字種)。

RLI-JEは、言語、エンコーディング、文字種の検出に N-gram アルゴリズムを使用します。155個の組込みプロファイルは、言語、エンコーディング、文字種の最も使用頻度の高い quad-gram (4 つの連続バイト) を含んでいます。入力テキストに対して検出を実行する際に、同様の N-gram プロファイルがそのデータに基づいて作成されます。入力プロファイルは、すべての組込みプロファイルと比較され、両プロファイル間のベクトル距離が計算されます。次に、入力プロファイルとのベクトル距離が最も近いものから順に、組込み前プロファイルが返されます。

短文解析.　短文解析には、ルールとモデルを駆使したアルゴリズムを用意しています。デフォルトでは短文解析のモデルをディスクから読み込みますが、LanguageIdentifierBuilder#useModelsInJar(boolean)オプションでJARファイルから読み込むことも可能です。

標準解析モデルおよび短文解析モデルで処理できる言語、エンコーディング、文字体系の一覧はRLI 言語　[11] をご覧ください。

第2章 RLI-JEの使い方

RLI-JEは単一言語の入力だけでなく、複数言語を含む入力データの言語も判別することができます。REX-JEを使うにはJava SDK 1.7以降が必要です。

2.1. 単一言語の判別

ここで紹介するコードには以下のインポートが必要です。特に断りがない限り、すべてのクラスは com.basistech.rosette.languageidentifier パッケージに含まれています。

```
import com.basistech.rosette.dm.AnnotatedText;
import com.basistech.rosette.dm.LanguageDetection;
import com.basistech.util.LanguageCode;
import com.basistech.util.ISO15924;
```

1. LanguageIdentifierBuilderを作成します。

   ```
   LanguageIdentifierBuilder liBuilder =
       new LanguageIdentifierBuilder(new File("path/to/rliRootDirectory"));
   ```

   ライセンスファイルをデフォルト位置(path/to/rliRootDirectory/licenses/rlp-license.xml)に置かなかった場合，license メソッドでライセンスを指定します。

2. (オプション) オプション [8] を設定します。

3. 入力テキストの言語判別を粉うには、まずAnnotatorを生成します:
   com.basistech.rosette.languageidentifier.LanguageIdentificationAnnotator
   はcom.basistech.rosette.dm.Annotator インターフェースを実装します。

   ```
   LanguageIdentificationAnnotator langAnnotator =
       liBuilder.buildSingleLanguageAnnotator();
   ```

   注記: 複数言語を含む入力テキストを処理するには言語領域の判別 [4] をご覧ください。

4. 入力テキストに注釈を付けるにはlangAnnotatorを使います。com.basistech.rosette.dm.AnnotatedText オブジェクトが返ります。

   ```
   AnnotatedText results =
       langAnnotator.annotate("This is the cereal shot from guns.");
   ```

5. AnnotatedText オブジェクトは、全入力テキストに対してLanguageDetection オブジェクトを取得するメソッドを含んでいます。

   ```
   LanguageDetection langDetect = results.getWholeTextLanguageDetection();
   ```

6. LanguageDetection オブジェクトは LanguageDetection.DetectionResult オブジェクトの一覧(信頼度スコア順)を含みます。

   ```
   List<LanguageDetection.DetectionResult> detectResults =
       langDetect.getDetectionResults();
   LanguageDetection.DetectionResult bestResult = detectResults.get(0);
   LanguageCode lang = bestResult.getLanguage();
   ISO15924 script =  bestResult.getScript();
   String encoding = bestResult.getEncoding();
   double confidence = bestResult.getConfidence();
   ```

RawDataの処理。 ここまでの処理はJava String (UTF-16)の入力を前提としています。UTF-8以外の入力、またはエンコーディング不明の入力を処理するにはcom.basistech.rosette.dm.RawData でannotatorのannotationの入力としてオブジェクトを使います。RLI-JEは言語、文字種、エンコーディングを判別します。サンプルの詳細はサンプル: sample.IdentifyExample [4] をご覧ください。

短文解析。 RLI-JEは、特殊な処理で短文判別精度を向上しています。com.basistech.rosette.languageidentifier.LanguageIdentifierBuilder shortStringThreshold()で短文の閾値を操作できます。デフォルトは0です(OFFの意)。例えば閾値を10と設定すると、10文字以内の文字列をすべて短文として処理します。

前述したようにRLI-JEでは、特殊なルールとモデルを駆使し、短文を処理します。languageHintと言語ウェイト以外、通常のoptions [8] は適用できません。入力はUTF-16形式の文字列です。短文の処理では他のエンコーディングには対応していません。短文の処理がON(shortStringThreshold > 0)で、入力がUTF-16以外の時、RLI-JEはUnsupportedOperationExceptionを返します。

短文の処理では、 LanguageDetection の結果を信頼度順に出力します。出力の各行には言語、文字体系、エンコーディング(UTF-16BE)、信頼度が表示されます。

例えばshortStringThresholdを7 に設定して**温州市委常委**を処理するとRLI-JEは以下の出力結果を返します。

```
Language(Script)/Encoding Confidence
Chinese(Hani)/UTF-16BE 0.924089
Japanese(Jpan)/UTF-16BE 0.075911
```

RLI-JEで処理できる短文の言語はRLI 言語 [11] をご覧ください。

短文解析で言語判別ができない場合、信頼度が1.0のUnknown (xxx) が返ります。

2.2. 言語領域の判別

多言語からなるテキストの場合、Annotatorは各言語領域に対しLanguageDetection オブジェクトを出力します。

1. LanguageIdentifierBuilder で 、言語領域の判別用にAnnotatorをビルドします。

```
Annotator regionAnnotator = new LanguageIdentifierBuilder(new File(rootDirectory))
                        .buildLanguageRegionAnnotator();
```

2. Annotator で 、言語領域の結果を含むAnnotatedTextオブジェクトをビルドします。

```
AnnotatedText lrResults = regionAnnotator.annotate(lrString);
```

3. 以下に各言語領域に対する判別言語などの属性を取得する例を示します。

```
for (LanguageDetection languageDetection
        : lrResults.getLanguageDetectionRegions()) {
            System.out.format("Region from %3d to %3d%n",
                    languageDetection.getStartOffset(),
                    languageDetection.getEndOffset());
            LanguageDetection.DetectionResult result =
                    languageDetection.getDetectionResults().get(0);
            System.out.format("%15s %15s %g%n",
                    result.getLanguage().languageName(),
                    result.getScript(),
                    result.getConfidence());
}
```

動作の詳細. Annotatorはまず文と文の区切りと文字体系の領域を判別します。各文字体系領域は、言語領域候補となります。評価が曖昧な場合は、文字体系領域内の各文章を評価します。文字体系が変わらない限り、文中言語領域は検出されません。

2.3. サンプル

サンプルプログラム (sample.IdentifyExample) では、単一言語や複数言語からなるドキュメントの言語、文字体系、エンコーディング判別の基本的な使い方を示します。

サンプルプログラムのコンパイルと実行には、Apache Antを使います。Antのビルドスクリプトは**RLP**ライセンスが **RLI-ROOT-DIRECTORY**/licenses/rlp-license.xmlにあることを想定しています。rlp-license.xmlを当該ディレクトリーにコピーしてからサンプルプログラムを実行してください。Antをパス変数に追加し、rli-je-7.15.0.c58.2/samples/rli-jeディレクトリーから次のようにAntを実行します:

```
ant run
```

サンプルは次の4つの処理を行います:

1. 単一言語のテキストファイル（example.txt）を標準解析（shortStringThreshold = 0）でraw テキスト（バイトの配列）として処理し、結果をコンソールに信頼度順に5つ出力します。(最も信頼度の高いものは、プロファイルデータとの差異が最も小さいものとなります。）各結果は言語、エンコーディング、信頼度を含みます。

2. 同じファイルをshortStringThreshold = 1でraw textとして処理し、UnsupportedOperationExceptionが返ります。RLI-JEはバイト配列内のテキスト長が不明なため、短文解析は行えません。

3. 短文のファイル(short.txt）からバイト配列を UTF-8 の文字列に変換します。その際shortStringThreshold の値をテキスト長よりも大きい値に設定し、短文解析を行います。信頼度の最も高い言語が判別結果として返ります。短文解析のサンプルとして"We'll study Machiavelli" という文章が含まれています。これは短文解析では英語と判別されますが、標準解析ではイタリア語として返ります。

4. 複数言語を含むテキストファイル（language-region-sample.txt）を処理し、最適結果を各言語領域の言語をオフセット値と共に返します。

ソースコード: rli-je-7.15.0.c58.2/samples/rli-je/src/sample/IdentifyExample.java

```
package sample;

import com.basistech.rosette.dm.AnnotatedText;
import com.basistech.rosette.dm.Annotator;
import com.basistech.rosette.dm.LanguageDetection;
import com.basistech.rosette.dm.RawData;
import com.basistech.rosette.languageidentifier.LanguageIdentificationAnnotator;
import com.basistech.rosette.languageidentifier.LanguageIdentifierBuilder;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.List;

import static java.nio.file.Files.readAllBytes;
import static java.nio.file.Paths.get;

/**
 * Creates a LanguageIdentifier instance,  prints the results
 * of language detection on the bytes from example_text, and the results
 * of language region detection on the bytes from language-region-sample.txt.
 */
public final class IdentifyExample {

    private IdentifyExample() {
    }

    private static byte[] getBytes(String fileName) throws IOException {
        File inFile = new File(fileName);
        byte[] bytes;
        try (InputStream in = new FileInputStream(inFile)) {
            // Get the size of the file
            long length = inFile.length();

            // byte array cannot be made from long
            if (length > Integer.MAX_VALUE) {
                // the file is too large for reading in one go
                throw new RuntimeException("File too large for single read");
            } else {
                bytes = new byte[(int) length];
            }

            // read in the bytes
            in.read(bytes);
        }
        return bytes;
    }


    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: IdentifyExample RLI-ROOT-DIRECTORY");
            System.exit(1);
```

```
            return;
        }
        IdentifyExample id = new IdentifyExample();
        id.run(args[0]);
    }

    public void run(String rootDirectory) throws Exception {
        // 1. Process example.txt as RawData with standard analysis: shortStringThreshold = 0.
        //    (shortStringThreshold = 0)
        // get the bytes from the example.txt file
        byte[] bytes = getBytes("example.txt");

        LanguageIdentificationAnnotator annotator = new LanguageIdentifierBuilder(new File(rootDirectory))
                .buildSingleLanguageAnnotator();
        // Process the monolingual text file (example.txt) as Raw Data;  shortStringThreshold is equal to 0,
        // so shortString language detection is inactive.
        try {
            RawData input = new RawData(ByteBuffer.wrap(bytes),
                    new HashMap<String, List<String>>());
            AnnotatedText results = annotator.annotate(input);

            // print the language detection results.
            System.out.println("1. RawData, shortStringThreshold = 0: Language Detection for example.txt.");
            System.out.println("-------------------------------------------------------------------------");
            System.out.format("%15s %15s %15s %s%n",
                    "Language",
                    "Encoding",
                    "Script",
                    "Confidence");
            for (LanguageDetection.DetectionResult result
                    : results.getWholeTextLanguageDetection().getDetectionResults()) {
                System.out.format("%15s %15s %15s %g%n",
                        result.getLanguage().languageName(),
                        result.getEncoding(),
                        result.getScript(),
                        result.getConfidence());
            }
        } catch (RuntimeException e) {
            e.printStackTrace();
        }

        System.out.println();


        // 2. Run short string algorithm with RawData.
        annotator = new LanguageIdentifierBuilder(new File(rootDirectory)).shortStringThreshold(1)
                .buildSingleLanguageAnnotator();
        // Process the monolingual text file (example.txt) as RawData, with shortStringThreshold > 0,
        // which throws UnsupportedOperationException, because shortString detection does
        // not support RawData input (length of string content is unknown).
        try {
            // print the language detection results.
            System.out.println("2. RawData, shortStringThreshold > 0: Language Detection for example.txt.");
            System.out.println("-------------------------------------------------------------------------");
            RawData input2 = new RawData(ByteBuffer.wrap(bytes),
                    new HashMap<String, List<String>>());
            AnnotatedText results2 = annotator.annotate(input2);

            for (LanguageDetection.DetectionResult result
                    : results2.getWholeTextLanguageDetection().getDetectionResults()) {
                System.out.format("%15s %15s %15s %g%n",
                        result.getLanguage().languageName(),
                        result.getEncoding(),
                        result.getScript(),
                        result.getConfidence());
            }
        } catch (RuntimeException e) {
            System.out.println(e.getLocalizedMessage());
        }
        System.out.println();

        // 3. Process the short-string text file (short.txt) as string (Annotated Text)
        //    and report the best result. Also, illustrate the use of LanguageIdentifierBuilder.license(String)
        //    when one wants to pass in a license via the API.
        bytes = getBytes("short.txt");
        try {
            // Get the license in the root directory for illustration purposes.
            String xmlLicense =
                    new String(readAllBytes(get(rootDirectory, "licenses", "rlp-license.xml")), StandardCharsets.UTF_8);
            String text = new String(bytes, StandardCharsets.UTF_8);
            int threshold = text.length() + 1;
            annotator =
                    new LanguageIdentifierBuilder(new File(rootDirectory))
```

```
                        .license(xmlLicense)
                        .shortStringThreshold(threshold)
                        .buildSingleLanguageAnnotator();

            // Alternatively, use the models bundled in the JAR.
            /*
            annotator =
                    new LanguageIdentifierBuilder(xmlLicense)
                            .useModelsInJar(true)
                            .shortStringThreshold(threshold)
                            .buildSingleLanguageAnnotator();
            */

            System.out.format("%s%d%s%n",

                    "3. String from file, shortStringThreshold = ",
                    threshold,
                    ": the best result with short-string analysis of short.txt.");
            System.out.println("--------------------------------------------------------------------------");
            System.out.format("%15s %15s %15s %s%n",
                    "Language",
                    "Encoding",
                    "Script",
                    "Confidence");
            AnnotatedText shortStringText = annotator.annotate(text);
            // Get the best result.
            LanguageDetection.DetectionResult result =
                shortStringText.getWholeTextLanguageDetection().getDetectionResults().get(0);
            System.out.format("%15s %15s %15s %g%n",
                        result.getLanguage().languageName(),
                        result.getEncoding(),
                        result.getScript(),
                        result.getConfidence());

        } catch (RuntimeException e) {
            e.printStackTrace();
        }
        System.out.println();

        // 4 Process a multilingual text file (language-region-sample.txt) and reports
        //   the best result for each region with regions offsets.
        try {
        /* now language regions */
            System.out.println("4. String Language Regions from language-region-sample.txt.");
            System.out.println("-----------------------------------------------------------------------------");
            Annotator regionAnnotator = new LanguageIdentifierBuilder(new File(rootDirectory))
                    .buildLanguageRegionAnnotator();
            byte[] lrData = getBytes("language-region-sample.txt");
            String lrString = new String(lrData, StandardCharsets.UTF_8);
            AnnotatedText lrResults = regionAnnotator.annotate(lrString);
            // print the best language detection result for each language region.
            for (LanguageDetection languageDetection
                    : lrResults.getLanguageDetectionRegions()) {
                System.out.format("Region from %3d to %3d%n",
                        languageDetection.getStartOffset(),
                        languageDetection.getEndOffset());
                LanguageDetection.DetectionResult result =
                        languageDetection.getDetectionResults().get(0);
                System.out.format("%15s %15s %g%n",
                        result.getLanguage().languageName(),
                        result.getScript(),
                        result.getConfidence());
            }
        } catch (RuntimeException e) {
            e.printStackTrace();
        }
    }
}
```

出力:

```
    [java] 1. RawData, shortStringThreshold = 0: Language Detection for example.txt.
    [java] ---------------------------------------------------------------------
    [java]        Language        Encoding          Script Confidence
    [java]         English        US-ASCII            Latn 0.0202112
    [java]        Romanian        US-ASCII            Latn 0.00291713
    [java]           Uzbek        US-ASCII            Latn 0.00282640
    [java]       Norwegian        US-ASCII            Latn 0.00273027
    [java]        Estonian        US-ASCII            Latn 0.00265288
    [java]
    [java] 2. RawData, shortStringThreshold > 0: Language Detection for example.txt.
    [java] ---------------------------------------------------------------------
    [java] shortStringThreshold = 1, but using the short string detection algorithm on RawData is not supported.
```

```
    [java]
    [java] 3. String from file, shortStringThreshold = 25: the best result with short-string analysis of short.txt.
    [java] ------------------------------------------------------------------------
    [java]        Language        Encoding          Script Confidence
    [java]         English        UTF-16BE            Latn 0.388209
    [java]
    [java] 4. String Language Regions from language-region-sample.txt.
    [java] ------------------------------------------------------------------------
    [java] Region from   0 to 2626
    [java]         Spanish           Latn 0.0133289
    [java] Region from 2626 to 3690
    [java]          French           Latn 0.0197861
    [java] Region from 3690 to 7834
    [java]          German           Latn 0.00932320
```

入力サンプルは任意に書き換え、Antを実行して解析できます。

rli-je-7.15.0.c58.2/samples/rli-je/build ディレクトリーに生成されたサンプルビルドは、次のようにして削除できます:

```
ant clean
```

2.4. オプション

標準解析には、いくつかのオプションが用意してあります。shortStringThreshold [9] を設定すると短文の処理が可能になりますが、それ以外の殆どのオプションは無効になります。

オプションの詳細はJava API のcom.basistech.rosette.languageidentifier.LanguageIdentifierBuilderをご覧ください。

- languageHint: 言語ヒントを与えます

  com.basistech.rosette.util.LanguageCode 定数で言語を示します。ウェイトは 1 から 99の浮動小数点です(デフォルトは1.0)。標準解析と短文解析では言語ヒントの扱いが異なります。

  本オプションは廃止予定で、languageWeightAdjustmentに置き換わりました。

  標準解析.　このヒントによって、指定されたウェイトの分（パーセント）、指定された言語プロファイルと入力プロファイルの距離が短縮されます。例えばデフォルトのウェイトでは、LanguageCode.DUTCHはオランダ語のプロファイルからの距離を1.0%短縮します。またLanguageCode.GERMANは、ウェイトが50.005の時、ドイツ語のプロファイルからの距離を50.005%短縮します。言語が分かっている可能性が非常に高い場合には、ヒントのウェイトを大きくして言語検出を抑制し、エンコーディングと文字体系の検出のみを実行することができます。

  短文解析.　短文解析では、ヒントのウェイトは標準解析以上に動作します。信頼度の増加率は次の計算式になります: 100/(100 - 重み)。したがってフランス語のウェイトが 50% の時、信頼度は2倍、 75%では4倍になります。

- encodingHint: エンコーディングのヒントを与えます。

  com.basistech.rosette.util.EncodingCode でエンコーディングを指定します。ウェイトは 1 から 100の浮動小数点です(デフォルトは1.0)。このヒントによって、指定されたウェイトの分（パーセント)、指定されたエンコーディング・プロファイルと入力プロファイルの距離が短縮されます。例えばデフォルトのウェイトでは、EncodingType.Ascii は入力プロファイルからの距離を1.0%短縮します。ウェイトが100の時、ヒントで与えたエンコーディングが判別時に使われます。

  なお本オプションは廃止予定です。

- 解析に必要な、空白以外の最低文字数を設定します。

  入力データの有効文字数が最小文字数より小さい場合、解析は行われません。値の範囲は、整数 1 以上です。

- profileDepth: プロファイルの深さを指定します。

  プロファイルの深さは、入力プロファイルで使用されるｎグラムの最大数です。深さが 100 の時、上位 100 の頻度の n グラムが入力プロファイルに組み込まれます。深さを小さくすると判別速度は向上しますが、判別精度は下がります。値の範囲は、整数 1 以上です。

- languageWeightAdjustment: 言語のウェイトを調整します。

  言語のウェイトを減少することで、入力に含まれる他の言語候補を検出します。languageWeightAdjustmentメソッドのweight引数は、言語ウェイトの0 - 100(パーセント)の範囲の数字です。設定値が70の時、言語ウェイトはデフォルト値より70%減少します。

  例えば英語のドキュメントに一部ドイツ語が含まれているようなデータで、両方の言語を検出するには、英語のウェイトを減らし、ドイツ語のウェイトを増加します。

- ambiguityThreshold: 曖昧度の閾値を設定します。

  入力プロファイルと候補の内蔵プロファイルの距離が、ambiguityThreshold/100 * bestProfileDistance の値よりも小さい時、判別結果は「曖昧」となります。bestProfileDistance は最適なプロファイルからの距離を表します。閾値が0 の時は「曖昧」な結果は出力されません。閾値が100 の時、すべての結果は「曖昧」となります。値は 1 から 100 の浮動小数点です。

- invalidityThreshold: 判別結果の「有効」「無効」の閾値を設定します。

  入力プロファイルの距離が invalidityThreshold/100 * maximumProfileDistance よりも小さい時、判別結果は「有効」となります。maximumProfileDistanceは入力テキストの持つプロファイルからの距離の最大値になります。閾値が0 の時、すべての結果は「無効」となり、閾値が100の時、すべての結果hは「有効」となります。値は 1 から 100 の浮動小数点です。

- minRegionLength: 言語領域の長さの最小値(文字体系の領域内の文字数)を設定します。本設定は言語領域の検出にのみ使われます。

- maxRegionLength: 言語領域の長さの最大値を設定します。本設定は言語領域の検出にのみ使われます。

- shortStringThreshold: 短文用に文字数の閾値を設定します。

  デフォルトでは0(短文解析はOFF)です。短文解析を行うには、この値を正の整数(6など)にします。入力の文字数が本設定よりも少ない時、短文解析 [3] が行われます。

  注記: 短文解析にはlanguageHintと言語ウェイトのみが使えます。その他のオプション [8] は無視されます。

- useModelsInJar: ファイルシステムのモデルの代わりに、JARファイルに含まれる短文解析モデルを使用します。本オプションは短文解析にのみ有効です。

- uniqueLanguages: 文字体系およびエンコーディングを考慮せずに言語判別を行います。

  簡体字中国語と繁体字中国語のように、同一言語で文字体系の異なる言語では、デフォルトで異なる言語が返ります。本オプションを設定すると、文字体系とエンコーディングが無視され、一つの言語(この場合は中国語)が返ります。

2.5. RLI-JE コマンドラインユーティリティー

RLI-JEにはコマンドラインユーティリティーも用意しています。すべての引数を確認するには以下を実行します。

```
bin/RLICmd -help
```

ファイルを処理して結果を見るには次を実行します。

```
bin/RLICmd -rootDirectory RLI-ROOT-DIRECTORY -in INPUT-FILE [Options]
```

RLI-ROOT-DIRECTORY はRLI-JEのルートディレクトリーになります。コマンドラインは、RLI-JEのルートからライセンスファイルを辿ります(RLI-ROOT-DIRECTORY/licenses/rlp-license.xml)。

INPUT-FILEは入力ファイルのパスです。短文解析では、入力ファイルはUTF-8のみに対応しています。

Options では使用するオプションを指定します。例えば多言語のデータから各言語の言語領域を検出するには -multilingual を指定します。短文解析を行うには、-maxShortStringLength n で指定します。n は短文の閾値になります。

出力結果は次のようになります。

言語(文字体系)/エンコーディング 信頼度

対応している言語-文字体系-エンコーディングの一覧を見るには、次を実行します。

```
bin/RLICmd -rootDirectory RLI-ROOT-DIRECTORY -profileInfo
```

2.6. RLI-JE パフォーマンスの最適化

2.6.1. マルチスレッド

Annotator オブジェクトは一つのスレッドに対して一つ使用します。LanguageIdentifierBuilder オブジェクトは、Annotator作成にマルチスレッドで使用できます。ただし設定値はスレッドセーフではありませんので、Annotator作成時には設定できません。したがってマルチスレッド環境ではAnnotatorをまとめて作成するとAnnotator作成時のオーバーヘッドが減少できます。Annotatorの設定は、後で変更不可ですから、複数のAnnotatorの設定が必要な場合、それぞれ個別に保存します。

2.6.2. メモリー

RLI-JEのJavaヒープサイズは150MB以上必要です。-Xmx の設定は (50 + 100 × [annotatorのスレッド数])MB で最適値が得られます。メモリーが限られている場合、メモリー値の半分に設定します。これはすべての言語、および短文解析でも同様です。

付録 A. RLI-JEの対応言語

RLI-JEの標準解析で対応する言語、文字体系、エンコーディングの一覧を次の表に示します。短文解析 [3] では、UTF-16BE (Java strings)以外のエンコーディングには対応していません。

言語 (ISO 639-3)	文字体系 (ISO 15924)	短文解析	エンコーディング
アルバニア語 (sqi)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
アラビア語 (ara)	Arabic (Arab)	✓	ISO-8859-6, UTF-16BE, UTF-16LE, UTF-8, windows-1256, windows-720
アラビア語 (ara)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252, windows-1256
ベンガル語 (ben)	Bengali (Beng)		ISCII-Bengali, UTF-16BE, UTF-16LE, UTF-8
ブルガリア語 (bul)	Cyrillic (Cyrl)		ISO-8859-5, KOI8-R, UTF-16BE, UTF-16LE, UTF-8, windows-1251
カタロニア語 (cat)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
中国語 (zho)	Han, Simplified (Hans)	✓[a]	GB18030, GB2312, HZ-GB-2312, ISO-2022-CN, UTF-16BE, UTF-16LE, UTF-8
中国語 (zho)	Han, Traditional (Hant)	✓[a]	Big5, UTF-16BE, UTF-16LE, UTF-8
クロアチア語 (hrv)	Latin (Latn)		ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
チェコ語 (ces)	Latin (Latn)	✓	ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
デンマーク語 (dan)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
オランダ語 (nld)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
英語-大文字 (uen[b])	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
英語 (eng)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
エストニア語 (est)	Latin (Latn)		ISO-8859-13, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1257
フィンランド語 (fin)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
フランス語 (fra)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
ドイツ語 (deu)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
ギリシャ語 (ell)	Greek (Grek)	✓	ISO-8859-7, UTF-16BE, UTF-16LE, UTF-8, windows-1253
グジャラート語 (guj)	Gujarati (Gujr)		ISCII-Gujarati, UTF-16BE, UTF-16LE, UTF-8
ヘブライ語 (heb)	Hebrew (Hebr)	✓	ISO-8859-8, UTF-16BE, UTF-16LE, UTF-8, windows-1255
ヒンズー語 (hin)	Devanagari (Deva)		ISCII-Devanagari, UTF-16BE, UTF-16LE, UTF-8
ハンガリー語 (hun)	Latin (Latn)	✓	ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
アイスランド語 (isl)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
インドネシア語 (ind)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
イタリア語 (ita)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
日本語 (jpn)	Japanese [Han + Hiragana + Katakana] (Jpan)	✓	EUC-JP, ISO-2022-JP, Shift_JIS, Shift_JIS-2004, UTF-16BE, UTF-16LE, UTF-8

言語 (ISO 639-3)	文字体系 (ISO 15924)	短文解析	エンコーディング
日本語 (jpn)	Katakana (Kana)		EUC-JP, Shift_JIS, Shift_JIS-2004, UTF-16BE, UTF-16LE, UTF-8
カンナダ語 (kan)	Kannada (Knda)		ISCII-Kannada, UTF-16BE, UTF-16LE, UTF-8
朝鮮語 (kor)	Korean [Hangul + Han] (Kore)	✓[c]	EUC-KR, ISO-2022-KR, UTF-16BE, UTF-16LE, UTF-8
クルド語 (kur)	Arabic (Arab)		UTF-16BE, UTF-16LE, UTF-8, windows-1256
クルド語 (kur)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252, windows-1256
ラトビア語 (lav)	Latin (Latn)		ISO-8859-13, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1257
リトアニア語 (lit)	Latin (Latn)		ISO-8859-13, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1257
マケドニア語 (mkd)	Cyrillic (Cyrl)		ISO-8859-5, UTF-16BE, UTF-16LE, UTF-8, windows-1251
マレー語 (msa)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
マラヤーラム語 (mal)	Malayalam (Mlym)		ISCII-Malayalam, UTF-16BE, UTF-16LE, UTF-8
ノルウェー語 (nor)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
ペルシャ語 (fas)	Arabic (Arab)	✓	UTF-16BE, UTF-16LE, UTF-8, windows-1256
ペルシャ語 (fas)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252, windows-1256
ポーランド語 (pol)	Latin (Latn)		ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
ポルトガル語 (por)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
パシュトゥ語 (pus)	Arabic (Arab)	✓	UTF-16BE, UTF-16LE, UTF-8, windows-1256
パシュトゥ語 (pus)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252, windows-1256
ルーマニア語 (ron)	Latin (Latn)	✓	ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
ロシア語 (rus)	Cyrillic (Cyrl)	✓	IBM866, ISO-8859-5, KOI8-R, UTF-16BE, UTF-16LE, UTF-8, windows-1251, x-mac-cyrillic
セルビア語 (srp)	Cyrillic (Cyrl)		ISO-8859-5, UTF-16BE, UTF-16LE, UTF-8, windows-1251
セルビア語 (srp)	Latin (Latn)		ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
スロバキア語 (slk)	Latin (Latn)		ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
スロベニア語 (slv)	Latin (Latn)		ISO-8859-2, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1250
ソマリ語 (som)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
スペイン語 (spa)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
スウェーデン語 (swe)	Latin (Latn)	✓	ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
タガログ語 (tgl)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252
タミール語 (tam)	Tamil (Taml)		ISCII-Tamil, UTF-16BE, UTF-16LE, UTF-8
テルグ語 (tel)	Telugu (Telu)		ISCII-Telugu, UTF-16BE, UTF-16LE, UTF-8
タイ語 (tha)	Thai (Thai)	✓	UTF-16BE, UTF-16LE, UTF-8, windows-874

言語 (ISO 639-3)	文字体系 (ISO 15924)	短文解析	エンコーディング
トルコ語 (tur)	Latin (Latn)	✓	ISO-8859-9, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1254
ウクライナ語 (ukr)	Cyrillic (Cyrl)		ISO-8859-5, KOI8-R, UTF-16BE, UTF-16LE, UTF-8, windows-1251
ウルドゥ語 (urd)	Arabic (Arab)		UTF-16BE, UTF-16LE, UTF-8, windows-1256
ウルドゥ語 (urd)	Latin (Latn)		ISO-8859-1, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1252, windows-1256
ウズベク語 (uzb)	Cyrillic (Cyrl)		ISO-8859-5, KOI8-R, UTF-16BE, UTF-16LE, UTF-8, windows-1251
ウズベク語 (uzb)	Latin (Latn)		US-ASCII, UTF-16BE, UTF-16LE, UTF-8, windows-1251
ベトナム語 (vie)	Latin (Latn)		TCVN, US-ASCII, UTF-16BE, UTF-16LE, UTF-8, VIQR, VISCII, VNI, VPS
全言語 (xxx[b])[d]	[Any script]	✓	

[a]中国語の短文解析では漢字が返ります。

[b]特殊言語コード

[c]朝鮮語の短文解析ではハングルが返ります。

[d]言語判別不能時に返ります。