package com.twitter.tweetypie.tweettext

import scala.util.matching.Regex

/\*\*

\* Code used to convert raw user-provided text into an allowable form.

\*/

object Preprocessor {

import TweetText.\_

import TextModification.replaceAll

/\*\*

\* Regex for dos-style line endings.

\*/

val DosLineEndingRegex: Regex = """\r\n""".r

/\*\*

\* Converts \r\n to just \n.

\*/

def normalizeNewlines(text: String): String =

DosLineEndingRegex.replaceAllIn(text, "\n")

/\*\*

\* Characters to strip out of tweet text at write-time.

\*/

val unicodeCharsToStrip: Seq[Char] =

Seq(

'\uFFFE', '\uFEFF', // BOM

'\uFFFF', // Special

'\u200E', '\u200F', // ltr, rtl

'\u202A', '\u202B', '\u202C', '\u202D', '\u202E', // Directional change

'\u0000', '\u0001', '\u0002', '\u0003', '\u0004', '\u0005', '\u0006', '\u0007', '\u0008',

'\u0009', '\u000B', '\u000C', '\u000E', '\u000F', '\u0010', '\u0011', '\u0012', '\u0013',

'\u0014', '\u0015', '\u0016', '\u0017', '\u0018', '\u0019', '\u001A', '\u001B', '\u001C',

'\u001D', '\u001E', '\u001F', '\u007F',

'\u2065',

)

val UnicodeCharsToStripRegex: Regex = unicodeCharsToStrip.mkString("[", "", "]").r

/\*\*

\* Strips out control characters and other non-textual unicode chars that can break xml and/or

\* json rendering, or be used for exploits.

\*/

def stripControlCharacters(text: String): String =

UnicodeCharsToStripRegex.replaceAllIn(text, "")

val Tweetypie674UnicodeSequence: String =

"\u0633\u0645\u064e\u0640\u064e\u0651\u0648\u064f\u0648\u064f\u062d\u062e " +

"\u0337\u0334\u0310\u062e \u0337\u0334\u0310\u062e \u0337\u0334\u0310\u062e " +

"\u0627\u0645\u0627\u0631\u062a\u064a\u062e \u0337\u0334\u0310\u062e"

val Tweetypie674UnicodeRegex: Regex = Tweetypie674UnicodeSequence.r

/\*\*

\* Replace each `Tweetypie674UnicodeSequence` of this string to REPLACEMENT

\* CHARACTER.

\*

\* Apple has a bug in its CoreText library. This aims to prevent

\* ios clients from being crashed when a tweet contains the specific

\* unicode sequence.

\*/

def avoidCoreTextBug(text: String): String =

Tweetypie674UnicodeRegex.replaceAllIn(text, "\ufffd")

/\*\*

\* Replace each `Tweetypie674UnicodeSequence` of this string to a REPLACEMENT

\* CHARACTER, returns a TextModification object that provides information

\* to also update entity indices.

\*/

def replaceCoreTextBugModification(text: String): Option[TextModification] =

replaceAll(text, Tweetypie674UnicodeRegex, "\ufffd")

private val preprocessor: String => String =

((s: String) => nfcNormalize(s))

.andThen(stripControlCharacters \_)

.andThen(trimBlankCharacters \_)

.andThen(normalizeNewlines \_)

.andThen(collapseBlankLines \_)

.andThen(avoidCoreTextBug \_)

/\*\*

\* Performs the text modifications that are necessary in the write-path before extracting URLs.

\*/

def preprocessText(text: String): String =

preprocessor(text)

/\*\*

\* Replaces all `<`, `>`, and '&' chars with "&lt;", "&gt;", and "&amp;", respectively.

\*

\* The original purpose of this was presumably to prevent script injections when

\* displaying tweets without proper escaping. Currently, tweets are encoded before

\* they are stored in the database.

\*

\* Note that the pre-escaping of & < and > also happens in the rich text editor in javascript

\*/

def partialHtmlEncode(text: String): String =

PartialHtmlEncoding.encode(text)

/\*\*

\* The opposite of partialHtmlEncode, it replaces all "&lt;", "&gt;", and "&amp;" with

\* `<`, `>`, and '&', respectively.

\*/

def partialHtmlDecode(text: String): String =

PartialHtmlEncoding.decode(text)

/\*\*

\*

\* Detects all forms of whitespace, considering as whitespace the following:

\* This regex detects characters that always or often are rendered as blank space. We use

\* this to prevent users from inserting excess blank lines and from tweeting effectively

\* blank tweets.

\*

\* Note that these are not all semantically "whitespace", so this regex should not be used

\* to process non-blank text, e.g. to separate words.

\*

\* Codepoints below and the `\p{Z}` regex character property alias are defined in the Unicode

\* Character Database (UCD) at https://unicode.org/ucd/ and https://unicode.org/reports/tr44/

\*

\* The `\p{Z}` regex character property alias is defined specifically in UCD as:

\*

\* Zs | Space\_Separator | a space character (of various non-zero widths)

\* Zl | Line\_Separator | U+2028 LINE SEPARATOR only

\* Zp | Paragraph\_Separator | U+2029 PARAGRAPH SEPARATOR only

\* Z | Separator | Zs | Zl | Zp

\* ref: https://unicode.org/reports/tr44/#GC\_Values\_Table

\*

\* U+0009 Horizontal Tab (included in \s)

\* U+000B Vertical Tab (included in \s)

\* U+000C Form feed (included in \s)

\* U+000D Carriage return (included in \s)

\* U+0020 space (included in \s)

\* U+0085 Next line (included in \u0085)

\* U+061C arabic letter mark (included in \u061C)

\* U+00A0 no-break space (included in \p{Z})

\* U+00AD soft-hyphen marker (included in \u00AD)

\* U+1680 ogham space mark (included in \p{Z})

\* U+180E mongolian vowel separator (included in \p{Z} on jdk8 and included in \u180E on jdk11)

\* U+2000 en quad (included in \p{Z})

\* U+2001 em quad (included in \p{Z})

\* U+2002 en space (included in \p{Z})

\* U+2003 em space (included in \p{Z})

\* U+2004 three-per-em space (included in \p{Z})

\* U+2005 four-per-em space (included in \p{Z})

\* U+2006 six-per-em space (included in \p{Z})

\* U+2007 figure space (included in \p{Z})

\* U+2008 punctuation space (included in \p{Z})

\* U+2009 thin space (included in \p{Z})

\* U+200A hair space (included in \p{Z})

\* U+200B zero-width (included in \u200B-\u200D)

\* U+200C zero-width non-joiner (included in \u200B-\u200D)

\* U+200D zero-width joiner (included in \u200B-\u200D)

\* U+2028 line separator (included in \p{Z})

\* U+2029 paragraph separator (included in \p{Z})

\* U+202F narrow no-break space (included in \p{Z})

\* U+205F medium mathematical space (included in \p{Z})

\* U+2061 function application (included in \u2061-\u2064)

\* U+2062 invisible times (included in \u2061-\u2064)

\* U+2063 invisible separator (included in \u2061-\u2064)

\* U+2064 invisible plus (included in \u2061-\u2064)

\* U+2066 left-to-right isolate (included in \u2066-\u2069)

\* U+2067 right-to-left isolate (included in \u2066-\u2069)

\* U+2068 first strong isolate (included in \u2066-\u2069)

\* U+2069 pop directional isolate (included in \u2066-\u2069)

\* U+206A inhibit symmetric swapping (included in \u206A-\u206F)

\* U+206B activate symmetric swapping (included in \u206A-\u206F)

\* U+206C inhibit arabic form shaping (included in \u206A-\u206F)

\* U+206D activate arabic form shaping (included in \u206A-\u206F)

\* U+206E national digit shapes (included in \u206A-\u206F)

\* U+206F nominal digit shapes (included in \u206A-\u206F)

\* U+2800 braille pattern blank (included in \u2800)

\* U+3164 hongul filler (see UCD Ignorable\_Code\_Point)

\* U+FFA0 halfwidth hongul filler (see UCD Ignorable\_Code\_Point)

\* U+3000 ideographic space (included in \p{Z})

\* U+FEFF zero-width no-break space (explicitly included in \uFEFF)

\*/

val BlankTextRegex: Regex =

"""[\s\p{Z}\u180E\u0085\u00AD\u061C\u200B-\u200D\u2061-\u2064\u2066-\u2069\u206A-\u206F\u2800\u3164\uFEFF\uFFA0]\*""".r

/\*\*

\* Some of the above blank characters are valid at the start of a Tweet (and irrelevant at the end)

\* such as characters that change the direction of text. When trimming from the start

\* or end of text we use a smaller set of characters

\*/

val BlankWhenLeadingOrTrailingRegex: Regex = """[\s\p{Z}\u180E\u0085\u200B\uFEFF]\*""".r

/\*\*

\* Matches consecutive blanks, starting at a newline.

\*/

val ConsecutiveBlankLinesRegex: Regex = ("""\n(""" + BlankTextRegex + """\n){2,}""").r

val LeadingBlankCharactersRegex: Regex = ("^" + BlankWhenLeadingOrTrailingRegex).r

val TrailingBlankCharactersRegex: Regex = (BlankWhenLeadingOrTrailingRegex + "$").r

/\*\*

\* Is the given text empty or contains nothing but whitespace?

\*/

def isBlank(text: String): Boolean =

BlankTextRegex.pattern.matcher(text).matches()

/\*\*

\* See http://confluence.local.twitter.com/display/PROD/Displaying+line+breaks+in+Tweets

\*

\* Collapses consecutive blanks lines down to a single blank line. We can assume that

\* all newlines have already been normalized to just \n, so we don't have to worry about

\* \r\n.

\*/

def collapseBlankLinesModification(text: String): Option[TextModification] =

replaceAll(text, ConsecutiveBlankLinesRegex, "\n\n")

def collapseBlankLines(text: String): String =

ConsecutiveBlankLinesRegex.replaceAllIn(text, "\n\n")

def trimBlankCharacters(text: String): String =

TrailingBlankCharactersRegex.replaceFirstIn(

LeadingBlankCharactersRegex.replaceFirstIn(text, ""),

""

)

/\*\* Characters that are not visible on their own. Some of these are used in combination with

\* other visible characters, and therefore cannot be always stripped from tweets.

\*/

private[tweettext] val InvisibleCharacters: Seq[Char] =

Seq(

'\u2060', '\u2061', '\u2062', '\u2063', '\u2064', '\u206A', '\u206B', '\u206C', '\u206D',

'\u206D', '\u206E', '\u206F', '\u200C',

'\u200D', // non-printing chars with valid use in Arabic

'\u2009', '\u200A', '\u200B', // include very skinny spaces too

'\ufe00', '\ufe01', '\ufe02', '\ufe03', '\ufe04', '\ufe05', '\ufe06', '\ufe07', '\ufe08',

'\ufe09', '\ufe0A', '\ufe0B', '\ufe0C', '\ufe0D', '\ufe0E', '\ufe0F',

)

private[tweetypie] val InvisibleUnicodePattern: Regex =

("^[" + InvisibleCharacters.mkString + "]+$").r

def isInvisibleChar(input: Char): Boolean = {

InvisibleCharacters contains input

}

/\*\* If string is only "invisible characters", replace full string with whitespace.

\* The purpose of this method is to remove invisible characters when ONLY invisible characters

\* appear between two urls, which can be a security vulnerability due to misleading behavior. These

\* characters cannot be removed as a rule applied to the tweet, because they are used in

\* conjuction with other characters.

\*/

def replaceInvisiblesWithWhitespace(text: String): String = {

text match {

case invisible @ InvisibleUnicodePattern() => " " \* TweetText.codePointLength(invisible)

case other => other

}

}

}