Scalaの文字列処理

Day 7　字種と文字の正規化

Unicodeコードポイントのグループ分け

グループ分け	特徴
Unicodeスクリプト	全てのUnicodeコードポイントは単一のUnicodeスクリプトに割り当てられます。
Unicodeブロック	連続するUnicodeコードポイントの塊。
Unicodeカテゴリ	全てのUnicodeコードポイントはUnicodeカテゴリに割り当てられます。
Unicodeバイナリ・プロパティ	Unicodeプロパティのうち、バイナリ型で定義されているものです。

Unicodeスクリプト

Unicodeスクリプト	説明
Common	記号
Han	漢字
Hiragana	ひらがな
Katakana	カタカナ
Latin	半角英数（Latin-1）

Unicodeブロック

Unicodeブロック	説明
Basic Latin	半角英数（Latin-1）
CJK Unified Ideographs	表意文字（漢字、BMP上のみ）
Hiragana	ひらがな
Katakana	カタカナ

Unicodeカテゴリ

Unicodeカテゴリ	説明	サブカテゴリ
C	その他 (Other)	Cc | Cf | Cs | Co | Cn
L	アルファベット (Letter)	Lu | Ll | Lt | Lm | Lo
M	記号 (Mark)	Mn | Mc | Me
N	数字 (Number)	Nd | Nl | No
P	句読記号 (Punctuation)	Pc | Pd | Ps | Pe | Pi | Pf | Po
S	記号 (Symbol)	Sm | Sc | Sk | So
Z	区切り文字 (Separator)	Zs | Zl | Zp

Unicodeバイナリ・プロパティ

Unicodeバイナリ・プロパティ	説明
Alphabetic	アルファベット
Ideographic	表意文字（漢字など）
Punctuation	句読点
Digit	数字

java.lang.Characterクラスによる文字の字種情報の取得

文字の字種情報を取得する３つのメソッド：

- Character.getName
- Character.getType
- Character.getDirectionality

Character.getName

コードポイントがunassignedの場合はnull、それ以外は次の結果を返す

```
Character.UnicodeBlock.of(codePoint).toString().
replace('_', ' ') + " " +
Integer.toHexString(codePoint).toUpperCase(Locale.ENGLISH);
```

Character.getType

Charやコードポイントに対してUnicodeカテゴリを返す

Character.getDirectionality

文字の方向性（双方向文字タイプ）を取得するために使用

文字の方向性というのは、例えば、日本語の文字は「左から右に表示する」といった情報のこと

java.lang.Characterクラスによる文字の字種判定

Characterクラスのメソッドで字種の判定を行う

引数はCharでもコードポイント（Int）でも可

引数がCharだとCharにはBMP領域の文字しか格納できないので、補助文字は判定できません。

正規表現の字種によるマッチング

正規表現には字種に関したマッチングを行うために、POSIX文字クラスや定義済み文字クラス（Unicodeスクリプト、ブロック、カテゴリ、バイナリ・プロパティなど）が用意されている。

POSIX文字クラス (US-ASCIIのみ)

POSIX（Portable Operating System Interface）標準に従う文字クラスにすべてのASCII文字を表すASCII文字クラス

POSIX文字クラス	マッチ	
\p{ASCII}	すべてのASCII文字	[\x00-\x7F]
\p{Alpha}	英字	[A-Za-z]
\p{Digit}	10 進数字	[0-9]
\p{Punct}	句読文字	!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~のうちのひと

java.lang.Characterクラス (単純なjava文字タイプ)

文字クラス	マッチ
\p{javaLowerCase}	java.lang.Character.isLowerCaseと等価
\p{javaUpperCase}	java.lang.Character.isUpperCaseと等価
\p{javaWhitespace}	java.lang.Character.isWhitespaceと等価
\p{javaMirrored}	java.lang.Character.isMirroredと等価

Unicodeスクリプト、ブロック、カテゴリ、バイナリ・プロパティのクラス

Unicodeブロックには接頭辞"In"、バイナリ・プロパティには接頭辞"Is"をつけることで定義済み文字クラスとして正規表現で記述可能

文字クラス	マッチ
\p{IsLatin}	Latin 書体文字(Unicodeスクリプト)
\p{InGreek}	Greek ブロックの文字(Unicodeブロック)
\p{Lu}	大文字(Unicodeカテゴリ)
\p{IsAlphabetic}	英字(Unicodeバイナリ・プロパティ)
\p{Sc}	通貨記号(Unicodeカテゴリ)
\P{InGreek}	ギリシャ語ブロック以外の文字(否定)
[\p{L}&&[^\p{Lu}]]	大文字以外の文字(減算)

POSIX文字クラスと定義済の文字クラスの互換性

POSIX文字クラス	マッチ	
\p{ASCII}	すべてのASCII文字	[\x00-\x7F]
\p{Alpha}	英字	\p{IsAlphabetic}
\p{Digit}	10 進数字	\p{IsDigit}
\p{Punct}	句読文字	\p{IsPunctuation}

日本語の字種のマッチング(1/2)

字種	正規表現	範囲
ひらがな	\p{Hiragana}, \p{InHiragana}	[U+3040, U+309F]
カタカナ	\p{Katakana}, \p{InKatakana}	[U+30A0, U+30FF]
ローマ字（大文字）	\p{Upper}, \p{IsUppercase}, A-Z	[U+0041, U+005A]
ローマ字（小文字）	\p{Lower}, \p{IsLowercase}, a-z	[U+0061, U+007A]
アラビア数字	\d, \p{Digit}, \p{IsDigit}, 0-9	[U+0030, U+0039]

日本語の字種のマッチング(2/2)

字種・文字クラス	正規表現	範囲
漢字	\p{Han}	
CJK互換漢字	\p{InCJKCompatibilityIdeographs}	[U+F900, U+FAFF]
CJK統合漢字	\p{InCJKUnifiedIdeographs}	[U+4E00, U+9FFF]
CJK統合漢字拡張A	\p{InCJKUnifiedIdeographsExtensionA}	[U+3400, U+4DBF]
CJK統合漢字拡張B	\p{InCJKUnifiedIdeographsExtensionB}	[U+20000, U+2A6DF]
CJK統合漢字拡張C	\p{InCJKUnifiedIdeographsExtensionC}	[U+2A700, U+2B73F]
CJK統合漢字拡張D	\p{InCJKUnifiedIdeographsExtensionD}	[U+2B740, U+2B81F]

字種の変換

- case付きのアルファベットをlower case、title case、upper caseに揃える方法

- ひらがなからカタカナ・カタカナからひらがなに変換する方法

letter case

letter case	例	説明
lower case	abc	全部小文字
title case	Abc	先頭文字が大文字で残りは小文字 １文字で複数文字あるように見える文字には、 title caseを持っている文字があります。 例： upper case:「LJ」（U+01C7） title case:「Lj」（U+01C8） lower case:「 lj」（U+01C9）
upper case	ABC	全部大文字

文字のletter caseの変換

Charやコードポイントのletter caseを
java.lang.CharacterクラスのtoUpperCase、
toTitleCase、toLowerCaseメソッドでいずれか一方
に揃える

Character.toLowerCase(upperCaseChar)

Character.toLowerCase(upperCaseCodePoint)

文字列のletter caseの変換

StringのletterをtoUpperCase、toLowerCaseメソッドで一方に揃える

string.toLowerCase

文字列の１文字目を大文字にする

string.capitalize

ひらがなとカタカナの相互変換（自作）

自作のJapaneseCharacterCaseConverterは、ひらがなとカタカナのコードポイントの差分を利用してひらがなとカタカナの相互変換を実行

JapaneseCharacterCaseConverter.convertKatakana2Hiragana(string).get.toString

JapaneseCharacterCaseConverter.convertHiragana2Katakana(string).get.toString

文字の正規化

半角カナ文字「ｱ」と全角カナ文字「ア」のように、言語処理で等価に扱いたい文字が存在します。どの文字とどの文字を等価に扱いたいかは厳密にはアプリケーションによって異なりますが、言語処理をする上では、このような等価性は前処理でいずれかの文字に揃えておくとアルゴリズムの複雑さを大幅に減らすことができます。この前処理のことを文字の正規化と呼びます。

文字の等価性

文字の等価性：

- 正準等価性 …
  - 例：「が」＝「か」＋「゛」
  - 左辺から右辺への変換＝「分解」（1対1なので、変換可能）
  - 右辺から左辺への変換＝「合成」（1対1なので、変換可能）
- 互換等価性 …
  - 例１：半角カナ文字「ｱ」＝全角カナ文字「ア」
  - 例２：全角英字「Ｃ」＝半角英字「C」
  - 左辺から右辺への変換＝「分解」（多対1を許す。変換可能）
  - 右辺から左辺への変換＝「合成」（1対多を許すので、変換不可能）

EUC-JP/Shift-JISでの正規化

文字を全て２バイト文字に揃える正規化方法が一般的

ひらがなやカタカナだけでなく英数字も半角文字を全角文字に変換

２バイト文字に揃えておくとバイト数割る２で文字数が計測可能

Unicode正規化

４種類の正規化形式：

- NFD … 正準等価性によって分解
- NFC … 正準等価性によって分解され、 再度合成
- NFKD … 互換等価性によって分解
- NFKC … 互換等価性によって分解され、 正準等価性によって再度合成。言語処理の文字正規化で使用

オプション

Optionは値があるのかないのかわからない状態を表すもの

値をOptionで包むことで、その値がある（nullではない）場合はSome、ない（nullである）の場合はNoneという状態に移り、Someの場合は値を取り出せるという機構

文字列オプション（自作）

文字列処理においてStringはnullだけでなく空文字""も同時に排除したい場合がよくあるが、Optionでは空文字は排除されない。そこで、Optionのようにを包むことでnullと空文字を排除するためのStringOptionを自作した。

https://github.com/ynupc/scalastringcourseday7/blob/master/src/test/scala/text/StringOption.scala

正規化文字列（自作）

値からnullや空文字を排除することに加えて、値が正規化されていることを保証するために
NormalizedStringを自作した。

https://github.com/ynupc/scalastringcourseday7/blob/master/src/test/scala/text/NormalizedString.scala

句点による文分割と文の正規化（自作）

日本語テキストを句点「。」や「．」の区切りによる文（単文・重文・複文の文ではない）に分割し、文を正規化する方法

単純にNormalizedStringやUnicode正規化してしまうと、日本語の句点・読点がそれぞれラテン文字のピリオト・カンマに変換されてしまう問題を回避