Engineer

正規表現

文字列が一致する条件を表すパターン

どのようなものか

正規表現は、文字列の集合を表す短いパターンで、プログラムが、その集合に属するものを選び出すために使います。The Open GroupとIEEEが発行するPOSIX標準は、この章の冒頭で、文字列の集合から特定の文字列を選ぶ道具と説明しています。日常の会話や、多くの関数・ライブラリー名では、regexと略します。JavaScriptのRegExpオブジェクトやGoのregexpパッケージのように、regexpという綴りも使います。

パターン内のほとんどの文字は、その文字自体を表し、パターンの `a` は、対象の文字列の `a` に一致します。一部は特別です。ピリオドは任意の一文字に一致し、 `*` は直前のものを零回以上繰り返し、 `[0-9]` のような角括弧の式は、集合内の一文字に一致します。 `^` と `$` は、始まりと終わりへ結び付け、 `{4}` のような繰り返し回数の指定は、正確な繰り返し回数を求めます。そのため、 `^[0-9]{4}$` は、ちょうど四桁だけで、前後にほかの文字がない文字列を表します。標準自身の回数指定の例は、 `c{3}` を `abababccccccd` に適用するものです。七文字目から九文字目、つまり六つあるcの最初の三つに一致します。

方言は一つではありません。POSIX自体が、ほとんどのユーティリティーが使う基本正規表現（BRE）と、一部が代わりに使う拡張正規表現（ERE）の二つを定義します。Cの `regcomp()` と `regexec()` は、両方を提供します。差は、見た目だけではありません。EREでは、 `a+` は、一つ以上の `a` を意味します。BREでは、 `+` は普通の文字で、 `\+` が、一つ以上を意味するか、文字どおりのプラス記号を意味するかは、標準が各実装へ委ねています。

もう一つの大きい系統は、Perlから来ています。ECMAScript仕様は、JavaScriptの正規表現の形式と機能が、Perl 5の正規表現機能を基にすると明記します。Goのregexpパッケージは、Perl、Pythonなどと、おおむね同じ構文を受け付けると説明し、三つ目の仲間が、Pythonの `re` モジュールです。記法の大半は共通で

も、細部まで同じではありません。省略形の文字クラスは、その一例です。数字の `\d` や単語文字の `\w` は、どこでも同じように見えますが、何を含むかは、以下の活用例のとおり、エンジンとフラグに依存します。

概念を判断に生かす

「正規表現とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 3社の間で受け渡す代わりに、1つのチームがアイデアのスケッチから、お客様がログインして使える製品まで担当します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GIS地図アプリの開発を主導し売上を10倍に — 10x 地図製品の開発を主導して実現した売上成長

GIS地図アプリの開発を主導し売上を10倍に お問い合わせ →

なぜ重要なのか

正規表現は、テキスト処理には見えない多くのソフトウェアにも入っています。フォームの入力検証、ログ行から値を取り出すフィルター、Webサーバーのルーティング規則、テキストエディターの検索と置換などです。一つ一つが短いため、実際に何を受け付けるかを、読み飛ばしがちです。結果を決める問いは二つあり、どちらも、答えはエンジン次第です。複数の一致があるときにどれを選ぶかと、一致を見つけるまでに、どれだけ時間がかかり得るかです。

POSIXは、最左最長一致を選びます。文字列内で最も早い位置から始まる一致のうち、最長のものを採ります。標準の例では、BREの `bb*` は、 `abbbc` の二文字目から四文字目に、EREの `(wee|week)(knights|night)` は、 `weeknights` の全十文字に一致します。Perl系は違います。選択肢を書いた順に試し、一致に至る最初のものを採ります。Goのドキュメントは、最左優先と呼びます。そのため、同じ文字列に同じパターンを使っても、二つの答えが出ます。

```
$ echo ab | grep -oE 'a|ab'
ab
>>> re.search("a|ab", "ab").group()
'a'
> "ab".match(/a|ab/)[0]
'a'
```

Goは、両方の動作を提供します。 `Compile` 関数は最左優先を使い、 `CompilePOSIX` は、構文をPOSIX EREに制限し、最左最長へ切り替えます。それでも、部分式の一致結果を選ぶときは、パッケージ独自の規則を使います。ドキュ

メントは、POSIXの規則は、計算のコストが高すぎる上、明確に定義されてさえいないためだと説明しています。

二つ目の問いは、時間です。Pythonの `re` は、ほとんどのPerl系エンジンと同じく、バックトラッキングします。パターンの一部を先へ進められなくなると、戻って、それ以前の部分の別の一致方法を試します。Pythonのドキュメントは、`a*a` を `aaaa` に照合する例で説明します。`a*` が、まず四文字をすべて取り、最後の `a` を一致させるために、一文字を戻します。大半のパターンでは、負担は目立ちません。一部では、試す方法の数が入力長に対して指数関数的に増え、数十文字で、プロセッサーを数分間使い続ける場合があります。MITREは、非効率な正規表現の計算量としてCWE-1333に分類し、ReDoS（正規表現によるサービス拒否）や、破滅的バックトラッキングという別名も挙げています。Goのregexpパッケージは、逆の保証を軸に設計されました。ドキュメントは、一致処理が「入力サイズに対して線形の時間で実行されることを保証する」とし、ほとんどのオープンソース実装には、その保証がないと補足しています。

実際の活用例

デンマークとウクライナの顧客向けの登録フォームで、名の欄に文字だけを許したい場合を考えます。思い付くパターンは `[A-Za-z]+` で、Python 3.15では、次のように動きます。

```
>>> import re
>>> re.fullmatch(r"[A-Za-z]+", "Mia")
<re.Match object; span=(0, 3), match='Mia'>
>>> re.fullmatch(r"[A-Za-z]+", "Søren")
>>> re.fullmatch(r"[A-Za-z]+", "Олена")
>>> re.fullmatch(r"\w+", "Søren_42")
<re.Match object; span=(0, 8), match='Søren_42'>
>>> re.fullmatch(r"[^\W\d_]+", "Олена")
<re.Match object; span=(0, 5), match='Олена'>
>>> re.fullmatch(r"[^\W\d_]+", "Søren_42")
>>>
```

対話シェルでは、`None` を返す呼び出しは、何の行も出力しません。つまり、ASCIIの文字クラスは、コペンハーゲンとキーウで、ごく普通の名前であるSørenとОлена（Olena）を拒否します。`\w` へ変えると通りますが、Pythonの `\w` は、`str.isalnum()` が真になるすべての文字と、アンダースコアを含むため、`Søren_42` も通します。`[^\W\d_]` は、単語文字以外、十進数字、アンダースコアを除きます。十進数字に分類されない数値文字も一部許すため、数字を除いて文字だけを受け付ける厳密な検査ではありません。複数の語からなる名前には、

`[^\W\d_]+( [^\W\d_]+)*` が、 `Anne Sofie Østergaard` も `Олена Коваль` も受け付けます。それでも、ハイフンでつないだ `Anne-Marie` は拒否するため、業務が許す限り、名前の欄の制約は緩く保つ理由になります。POSIXのツールにも、別の形で同じ問題があります。標準は、POSIXロケール以外での `A-Z` のような範囲の意味を未規定とし、現在のロケールによって構成文字が決まる `[[:alpha:]]` クラスを提供します。

これは、一つのフォームだけの問題ではありません。このサイトの国際化とローカライズは、顧客が販売するすべての市場で、名前の並び順まで含め、自然に読めるソフトウェアを目指すサービスです。Sørenを拒否する名前の欄は、並べ替えに至るよりずっと前に、その目標を満たしません。

数字には、逆の問題があります。デンマークの郵便番号は四桁なので、 `\d{4}` は、 `2200` を安全に検査できそうに見えます。しかしPythonでは、文字列用のパターンの `\d` は、0から9だけでなく、Unicodeのあらゆる十進数字に一致し、アラビア・インド数字で書いた同じ郵便番号も受け付けます。

```
>>> s = "\u0662\u0662\u0660\u0660"
>>> print(s, int(s))
٢٢٠٠ 2200
>>> re.fullmatch(r"\d{4}", s)
<re.Match object; span=(0, 4), match='٢٢٠٠'>
>>> re.fullmatch(r"\d{4}", s, re.ASCII)
>>> re.fullmatch(r"[0-9]{4}", s)
>>>
```

JavaScriptは、逆の選択をしています。ECMAScript仕様は、 `\d` を、ちょうど0から9の十文字と定義するため、同じ文字列に対する `/^\d{4}$/.test(s)` は、Node.jsで `false` を返し、 `/^\p{Nd}{4}$/u` なら `true` を返します。ブラウザーとサーバーで別の規則を使うフォームは、同じ入力について判断が食い違います。Pythonの `int()` は、その文字列を2200と読めても、データベースの列、検索用のテーブル、別のサービスは、読めないかもしれません。どの数字を指すのかを決めます。ASCII数字だけなら、 `[0-9]` か `re.ASCII` フラグを使い、JavaScriptで、どの十進数字も受け入れたい場合は、 `\p{Nd}` と `u` フラグを使います。

問題が起きるところ

一文字ごとに、実行時間が倍になります。 一つの空白で区切る語を許すつもりの名前のパターンは、簡単に `([^\W\d_]+ ?)+` と書けます。正しい名前なら速く動きます。文字の連続の後ろに、受け付けない文字が来ると、諦めるまでに、文字

を語へ分けるあらゆる方法を試し、一文字ごとに方法の数が倍になります。ある一台のマシンのPython 3.15で、各長さを五回ずつ実行した最速値は、`fullmatch`を、文字の列の最後に`!`を付けた文字列へ実行した場合、次のとおりでした。

```
文字数    秒
    18    0.017
    20    0.067
    22    0.27
    24    1.0
    26    4.1
```

一文字で時間が倍になるため、二文字増えると四倍です。その倍率から外挿すると、三十文字で約一分、三十二文字で四分を超えます。CWE-1333が説明するのは、このケースです。バックトラッキングの試行が入力長に対して指数関数的に増え、入力は一致する必要がなく、十分に長くできることです。`[^\W\d_]+([^\W\d_]+)*`への書き換えは、入れ子の量指定子を使わず、同じ名前を表し、24文字の文字列を約一マイクロ秒で、百万文字の後ろに`!`を付けたものを約三十ミリ秒で拒否します。量指定子の入れ子をなくし、一致を調べる前に入力長へ上限を設け、信頼できない入力を扱うパターンは、Goのように線形時間を保証するエンジンで動かします。

値全体が一致すべき場所で、検索をします。 Pythonの`re.search`は、文字列内のどこかに一致すればよく、`re.match`が固定するのは先頭だけです。Python 3.15は、より明確な`prefixmatch`という名前を与え、以前の名前をソフト非推奨扱いにし、ほかの言語のmatchは、Pythonが従来searchと呼ぶものだと説明しています。そのため、`re.prefixmatch(r"\d{4}", "22000")`は最初の四桁で成功し、`re.search(r"\d{4}", "DK-2200")`は、文字列内の郵便番号を見つけます。`re.search(r"^\d{4}$", "2200\n")`さえ成功します。`$`は、末尾の改行の直前にも一致するためです。Python 3.4以来の`re.fullmatch`は、三つとも拒否します。パターンが値全体を表す場合は、`fullmatch`を使うか、`\A`と`\Z`で固定します。Python 3.14は、後者に`\z`という綴りも用意しています。

ドットの緩い許可リストを使います。 リダイレクトを自社サイトだけに許す検査を、`^https://www.example.com`と書く場合があります。これは、`https://www.example.com.evil.example/`を通します。パターンがホスト名の終わりまで達していないためです。また、エスケープしていない各ドットは、任意の一文字に一致するため、本当のホスト名のドット一つを、別の一文字に変えたものも

通します。`re.escape("www.example.com")`のようにドットをエスケープし、パターンを`/`で終えれば、両方を防げます。さらによいのは、URLを解析し、ホスト名全体を、期待する文字列と比較することです。

パターンを、別のエンジンへ複製します。 `grep -E`でテストしてPythonへ貼ったパターンは、`a|ab`の例のように、違う一致を選ぶ場合があります。一つのBREツールで動く`\+`も、別のツールでは、文字どおりのプラス記号かもしれません。Pythonでアラビア・インド数字を受け付ける`\d`も、JavaScriptでは拒否します。どのパターンも、実際に動かすエンジンとフラグでテストします。

グローバルフラグが、状態を覚えます。 JavaScriptの`g`か`y`フラグ付きの正規表現は、前回の一致の終了位置を`lastIndex`に保存し、次の検索を、そこから始めます。失敗したら、`lastIndex`を0へ戻します。パターンを定数に保持し、検証に使い回すと、呼び出すたびに答えが変わります。

```
> const re = /^\d{4}$/g;
undefined
> re.test("2200")
true
> re.test("2200")
false
> re.test("2200")
true
```

`g`フラグを、`test()`に使うパターンへ付けないか、毎回、呼び出す前に`lastIndex`を0にします。

見た目は同じでも、文字の表し方が違います。 Andréという名前は、合成済みのé一文字でも、普通のeの後ろに結合アクセントを置いても保存できます。画面では同じでも、POSIXは、一致が字形ではなく、文字の符号化に使うビットパターンに従うと明記し、Pythonの`re.search("André", "Andre\u0301")`は`None`を返します。文字列を`unicodedata.normalize("NFC", ...)`へ渡した後なら、見つけられます。一致を調べる前に、パターンと文字列を、同じ形式へ正規化します。Pythonでは`unicodedata.normalize`、JavaScriptでは`String.prototype.normalize`を使います。

参考資料

- The Open Group基本仕様、第8版：正規表現 — BREとERE、角括弧の式、文字クラスとロケール、最左最長の規則についてのPOSIXの定義です。

- regexpパッケージ（Go） — 線形時間の保証、最左優先と最左最長の違い、CompilePOSIXが部分式の一致結果でPOSIXと異なる理由です。
- re — 正規表現の操作（Python） — 構文、UnicodeとASCIIのクラス、バックトラッキングと独占的量指定子（possessive quantifier）、search、prefixmatch、fullmatchです。
- ECMAScript：テキスト処理（更新中のドラフト、ECMAScript 2027） — Perl 5を継承するJavaScriptの正規表現、`\d` の定義、Unicodeプロパティーエスケープ、lastIndexの使い方です。
- CWE-1333：非効率な正規表現の計算量 — 破滅的バックトラッキングとReDoSで、悪用できる条件、観測された例、緩和策を説明します。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「正規表現とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/regular-expression/