Engineer

Expression régulière

Motif définissant les chaînes qui correspondent

Ce que c'est

Une expression régulière est un motif court décrivant un ensemble de chaînes, pour qu'un programme puisse sélectionner celles qui en font partie. La norme POSIX, publiée par The Open Group et l'IEEE, ouvre son chapitre sur le sujet en décrivant les expressions régulières comme l'outil permettant de sélectionner certaines chaînes parmi un ensemble de chaînes de caractères. Dans le langage courant et dans de nombreux noms de fonctions et de bibliothèques, le terme se raccourcit en regex ou regexp, graphie de l'objet RegExp de JavaScript et du paquet regexp de Go.

La plupart des caractères d'un motif se représentent eux-mêmes : un `a` dans le motif correspond à un `a` dans le texte. Quelques-uns sont spéciaux. Un point correspond à n'importe quel caractère, `*` répète ce qui le précède zéro fois ou plus, une expression entre crochets comme `[0-9]` correspond à un caractère d'un ensemble, `^` et `$` attachent le motif au début et à la fin, et un intervalle comme `{4}` demande un nombre exact de répétitions. Ainsi, `^[0-9]{4}$` décrit toute chaîne composée d'exactement quatre chiffres, sans autre caractère avant ni après. L'exemple d'intervalle donné par la norme est `c{3}` appliqué à `abababccccccd` : il correspond aux septième à neuvième caractères, les trois premiers des six c.

Il existe plusieurs dialectes. POSIX en définit deux : les expressions régulières de base, BRE, utilisées par la plupart de ses utilitaires, et les expressions régulières étendues, ERE, utilisées par certains à leur place. Les fonctions C `regcomp()` et `regexec()` proposent les deux. La différence n'est pas cosmétique. Dans une ERE, `a+` signifie un ou plusieurs `a`. Dans une BRE, `+` est un caractère ordinaire, et la norme laisse chaque implémentation décider si `\+` signifie un ou plusieurs ou un signe plus littéral.

L’autre grande famille descend de Perl. La spécification ECMAScript indique que la forme et le fonctionnement des expressions régulières de JavaScript sont modelés sur ceux de Perl 5. Le paquet regexp de Go dit accepter globalement la même syntaxe que Perl, Python et d’autres langages, et le module `re` de Python est un troisième membre de cette famille. Ces moteurs partagent l’essentiel de leur notation, mais pas tous les détails. Les classes abrégées l’illustrent : `\d`, un chiffre, et `\w`, un caractère de mot, se ressemblent partout, mais leur contenu dépend du moteur et de ses options, comme le montre la partie pratique ci-dessous.

Utilisez le concept pour prendre une décision

Utilisez « Qu’est-ce qu’une expression régulière ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une seule équipe transforme une idée en un produit auquel vos clients peuvent se connecter, sans le faire passer entre trois prestataires.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Application GIS : revenus multipliés par 10 — 10x croissance du chiffre d'affaires portée par le produit cartographique

Application GIS : revenus multipliés par 10 Contactez-nous →

Pourquoi c’est important

Les expressions régulières se trouvent dans beaucoup de logiciels qui ne semblent pas relever du traitement de données textuelles : validation de champs de formulaire, filtres extrayant des valeurs des lignes de logs, règles de routage des serveurs web, recherche et remplacement dans les éditeurs. Chaque motif est court, ce qui facilite une lecture trop rapide de ce qu’il accepte réellement. Deux questions décident du résultat, et leurs réponses dépendent du moteur : quelle correspondance est choisie lorsque plusieurs sont possibles, et combien de temps sa recherche peut prendre.

POSIX choisit la correspondance la plus longue parmi celles situées le plus à gauche : parmi toutes celles qui commencent le plus tôt dans le texte, la plus longue gagne. Les exemples de la norme sont la BRE `bb*`, qui correspond aux deuxième à quatrième caractères de `abbbc`, et l’ERE `(wee|week)(knights|night)`, qui correspond aux dix caractères de `weeknights`. La famille Perl fonctionne autrement. Elle essaie les alternatives dans l’ordre où elles sont écrites et prend la première qui mène à une correspondance, ce que la documentation Go appelle leftmost-first. Le même motif appliqué au même texte peut donc donner deux réponses :

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

Go propose les deux comportements. Sa fonction `Compile` utilise la sémantique leftmost-first, tandis que `CompilePOSIX` limite la syntaxe aux ERE POSIX et passe à leftmost-longest. Même dans ce cas, le paquet garde sa propre règle de choix des sous-correspondances. Sa documentation explique pourquoi : la règle POSIX coûterait trop cher à calculer et n'est même pas clairement définie.

La seconde question est le temps. Le module `re` de Python, comme la plupart des moteurs de la famille Perl, revient en arrière. Lorsqu'une partie du motif ne peut plus avancer, le moteur recule et essaie une autre façon de faire correspondre ce qui précède. La documentation Python illustre l'idée avec `a*a` face à `aaaa` : `a*` prend d'abord les quatre lettres, puis en rend une pour que le `a` final puisse correspondre. Le coût est imperceptible pour la plupart des motifs. Pour certains, le nombre de possibilités à essayer croît exponentiellement avec la longueur de l'entrée, et une chaîne de quelques dizaines de caractères peut occuper un processeur pendant plusieurs minutes. MITRE répertorie cela comme CWE-1333, « complexité inefficace des expressions régulières », avec ReDoS, déni de service par expression régulière, et retour arrière catastrophique comme autres appellations. Le paquet regexp de Go a été conçu autour de la promesse inverse. Sa documentation garantit que la recherche « s'exécute dans un temps linéaire par rapport à la taille de l'entrée » et ajoute que la plupart des implémentations open source n'offrent pas cette garantie.

À quoi cela ressemble en pratique

Prenons un formulaire d'inscription pour des clients au Danemark et en Ukraine, avec un champ prénom censé n'accepter que des lettres. Le motif évident est `[A-Za-z]+` . Dans Python 3.15, il se comporte ainsi :

```
>>> 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")
>>>
```

Un appel qui renvoie `None` n'affiche aucune ligne dans le shell interactif. La classe ASCII refuse donc Søren et Олена (Olena), deux prénoms tout à fait ordinaires à Copenhague et à Kyiv. Passer à `\w` les accepte, mais en Python `\w` couvre tous les caractères pour lesquels `str.isalnum()` vaut vrai, plus le trait de soulignement : il accepte donc aussi `Søren_42`. La classe `[^\W\d_]` exclut les caractères qui ne sont pas des caractères de mot, les chiffres décimaux et le trait de soulignement. Elle autorise aussi certains caractères numériques non décimaux : ce n'est donc pas un test strictement limité aux lettres. Pour les noms de plusieurs mots, `[^\W\d_]+( [^\W\d_]+)*` accepte aussi bien `Anne Sofie Østergaard` que `Олена Коваль`. Ce motif refuse encore `Anne-Marie` avec son trait d'union, raison de garder un champ nom aussi permissif que le permettent les besoins métier. Les outils POSIX rencontrent le même problème sous une autre forme : la norme ne définit pas le sens d'un intervalle comme `A-Z` hors de la locale POSIX et propose la classe `[[:alpha:]]`, dont les membres sont définis par la locale courante.

Bien traiter ce point importe au-delà d'un formulaire. L'une des prestations de ce site, l'internationalisation et la localisation, vise des logiciels dont les textes se lisent naturellement dans chaque marché où le client vend, jusqu'à l'ordre de tri des noms. Un champ qui refuse Søren manque cet objectif bien avant tout tri.

Les chiffres posent la question inverse. Un code postal danois comporte quatre chiffres, donc `\d{4}` semble être un contrôle fiable pour `2200`. En Python, toutefois, `\d` dans un motif textuel correspond à tout chiffre décimal Unicode, pas seulement de 0 à 9, et accepte le même code postal écrit en chiffres arabo-indiens :

```
>>> 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 fait l'autre choix. La spécification ECMAScript définit `\d` comme exactement les dix caractères de 0 à 9 : `/^\d{4}$/.test(s)` renvoie donc `false` dans Node.js pour la même chaîne, tandis que `/^\p{Nd}{4}$/u` renvoie `true`. Un formulaire contrôlé dans le navigateur avec une règle et sur le serveur avec l'autre donnera des réponses différentes pour la même entrée. Le `int()` de Python lit la chaîne comme 2200, mais une colonne de base de données, une table de correspondance ou un autre service ne le fera pas nécessairement. Décidez quels chiffres sont attendus : `[0-9]` ou l'option `re.ASCII` si seuls les chiffres ASCII conviennent, et `\p{Nd}` avec l'option `u` en JavaScript si tous les chiffres décimaux sont acceptés.

Où les problèmes surviennent

Un motif dont le temps d'exécution double à chaque lettre. Un motif destiné à autoriser des mots séparés par un seul espace s'écrit facilement `([^\W\d_]+ ?)+`. Il est rapide sur un nom valide. Sur une suite de lettres suivie d'un caractère refusé, le moteur essaie toutes les façons de répartir les lettres en mots avant d'abandonner, et leur nombre double à chaque lettre. Sur une machine avec Python 3.15, le plus rapide de cinq essais à chaque longueur a donné les temps suivants pour `fullmatch` sur une chaîne de lettres suivie de `!` :

```
letters   seconds
     18     0.017
     20     0.067
     22     0.27
     24     1.0
     26     4.1
```

Chaque lettre double le temps, donc deux lettres supplémentaires le multiplient par quatre. En extrapolant ce facteur, trente lettres prendraient environ une minute et trente-deux lettres, plus de quatre minutes. C'est le cas décrit par CWE-1333 : les tentatives de retour arrière croissent exponentiellement avec l'entrée, celle-ci n'a pas besoin de correspondre et elle peut être assez longue. La réécriture `[^\W\d_]+( [^\W\d_]+)*` décrit les mêmes noms sans quantificateur imbriqué, rejette la chaîne de 24 lettres en environ une microseconde et un million de lettres suivies de `!` en environ trente millisecondes. Supprimez les quantificateurs imbriqués, plafonnez la longueur avant la recherche et exécutez les motifs recevant des entrées non fiables sur un moteur garantissant un temps linéaire, comme celui de Go.

Une recherche alors que toute la valeur doit correspondre. En Python, `re.search` accepte une correspondance n'importe où dans la chaîne et `re.match` ne l'ancre qu'au début. Python 3.15 lui donne le nom plus clair de `prefixmatch` et déconseille l'ancien de façon non contraignante, en rappelant que d'autres langages utilisent match pour ce que Python a toujours appelé search. Ainsi, `re.prefixmatch(r"\d{4}", "22000")` réussit sur les quatre premiers chiffres, et `re.search(r"\d{4}", "DK-2200")` trouve le code postal à l'intérieur de la chaîne. Même `re.search(r"^\d{4}$", "2200\n")` réussit, car `$` correspond aussi à la position précédant une nouvelle ligne finale. `re.fullmatch`, disponible depuis Python 3.4, refuse les trois. Utilisez `fullmatch` chaque fois qu'un motif décrit la valeur entière, ou ancrez-le avec `\A` et `\Z`, également écrit `\z` dans Python 3.14.

Une liste d'autorisation avec un point trop large. Un contrôle destiné à n'autoriser les redirections que vers le site de l'entreprise pourrait s'écrire `^https://www.example.com`. Il accepte `https://www.example.com.evil.example/`, car le motif s'arrête avant la fin du nom d'hôte. Chaque point non échappé correspond en outre à n'importe quel caractère : un hôte différant du véritable par un seul caractère à la place d'un point passe donc aussi. Échapper les points, comme le fait `re.escape("www.example.com")`, et terminer le motif par `/` ferme les deux brèches. Mieux encore, analysez l'URL et comparez son nom d'hôte entier à celui attendu.

Un motif copié entre moteurs. Un motif testé avec `grep -E` puis collé dans Python peut choisir une autre correspondance, comme l'illustre `a|ab`. Un motif fonctionnant avec `\+` dans un outil BRE peut signifier un plus littéral dans un autre. Le `\d` acceptant les chiffres arabo-indiens dans Python les refuse en JavaScript. Testez chaque motif dans le moteur qui l'exécutera, avec les options qu'il utilisera.

Une option globale qui se souvient. En JavaScript, une expression régulière avec l'option `g` ou `y` conserve dans `lastIndex` la position où s'est terminée sa dernière correspondance et commence la recherche suivante à cet endroit. Après un échec, elle remet `lastIndex` à 0. Un motif conservé dans une constante et réutilisé pour la validation répond alors différemment un appel sur deux :

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

```
false
> re.test("2200")
true
```

N'utilisez pas l'option `g` sur les motifs employés avec `test()`, ou remettez `lastIndex` à 0 avant chaque appel.

Un texte qui semble identique sans l'être. Le nom André peut être stocké avec un é précomposé ou avec un e simple suivi d'un accent sous forme de caractère combinant. Les deux paraissent identiques à l'écran, mais POSIX précise que la correspondance suit la représentation binaire utilisée pour encoder un caractère, pas son apparence graphique, et `re.search("André", "Andre\u0301")` renvoie `None` en Python. Après `unicodedata.normalize("NFC", ...)` sur le texte, il trouve le nom. Normalisez le motif et le texte sous la même forme avant la recherche, avec `unicodedata.normalize` en Python ou `String.prototype.normalize` en JavaScript.

Pour aller plus loin

- The Open Group Base Specifications Issue 8 : expressions régulières — définition POSIX : BRE et ERE, expressions entre crochets, classes de caractères et locales, et règle de la correspondance la plus longue la plus à gauche.
- Paquet regexp de Go — garantie de temps linéaire, correspondances leftmost-first et leftmost-longest, et raison pour laquelle CompilePOSIX s'écarte de POSIX pour les sous-correspondances.
- re — opérations sur les expressions régulières en Python — syntaxe, classes Unicode et ASCII, retour arrière et quantificateurs possessifs, ainsi que search, prefixmatch et fullmatch.
- ECMAScript : traitement de texte, brouillon évolutif ECMAScript 2027 — expressions régulières JavaScript : héritage de Perl 5, définition de `\d`, échappements de propriétés Unicode et utilisation de lastIndex.
- CWE-1333 : complexité inefficace des expressions régulières — retour arrière catastrophique et ReDoS : conditions rendant un motif exploitable, exemples observés et mesures d'atténuation.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'une expression régulière ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/regular-expression/