Regulární výrazy

PVA2 Programování a vývoj aplikací

Adam Fišer

Obsah

- Autor: Mirza Sisic

Co jsou to regulární výrazy?

- Regulární výrazy jsou způsob, jak popsat vzory v textu
- Konstrukce vychází z teorie formálních jazyků a překladačů.
- Jsou to řetězce, které popisují, jaký text hledáme
- Používají se v různých programovacích jazycích a nástrojích
- Například v JavaScriptu, Pythonu, PHP, nebo v textových editorech
- Základní syntax stejná ve většině jazycích, ale mohou se lišit v detailu
- Jsou velmi mocným nástrojem pro práci s textem
- Jsou také velmi těžké na pochopení a psaní
- Většina programátorů je neumí správně používat

Použití regulárních výrazů

- Vyhledávání textu
- Nahrazování textu
- Validace vstupů
  - Například validace emailu, hesla, telefonního čísla, atd.
- Extrahování dat
- Filtraci dat

Příklady použití

- Vyhledání všech emailů v textu
- Vyhledání všech URL v textu
- Vyhledání všech telefonních čísel v textu
- Kontrola, zda je heslo dostatečně silné
- Kontrola, zda je email validní
- Kontrola, zda je telefonní číslo validní

Pomocníci

- Regex101
- RegExr

Základní syntaxe

- V syntaxi regulárních výrazů má každý znak svůj speciální význam.
- Základní znaky se hledají přesně tak, jak jsou napsány.
- Kvantifikátory: `*` `+` `?` `{n}` `{n,}` `{n,m}`
- Skupiny: `()` `[]`
- Rozsahy: `a-z` `A-Z` `0-9`
- Speciální znaky: `.` `^` `$` `*` `+` `?` `\` `|`

Kvantifikátory

- Znak kvantifikátoru určuje, kolikrát se má znak před ním opakovat.
- Kvantifikátory se vždy vztahují k jednomu znaku nebo skupině znaků.
- `*` - nula nebo více opakování
- `+` - jedno nebo více opakování
- `?` - nula nebo jedno opakování
- `{n}` - přesně n opakování
- `{n,}` - n nebo více opakování
- `{n,m}` - od n do m opakování
- `*?` - hledá co nejmenší možné opakování

Příklady

```
1  a* - a, aa, aaa, aaaa, ...
2  a+ - aa, aaa, aaaa, ...
3  a? - a, aa
4  a{3} - aaa
5  a{3,} - aaa, aaaa, aaaaa, ...
6  a{3,5} - aaa, aaaa, aaaaa
7  a*? - a, aa, aaa, aaaa, ...
```

Zástupné znaky

- Zástupné znaky se používají pro hledání libovolného znaku nebo skupiny znaků.
- `.` - libovolný znak
- `[]` - libovolný znak z dané množiny
- `[^]` - libovolný znak, který není v dané množině
- `()` - skupina znaků
- `|` - logický OR, pravý `Alt` + `W`
- `\` - escape znak
- `^` - začátek řádku, pravý `alt` + `3`
- `$` - konec řádku, pravý `alt` + `ů`

Příklady

```
a.c - abc, acc, a1c, a#c, ...
[a-z] - libovolný znak z a-z
[^a-z] - libovolný znak, který není z a-z
^a - začíná na a
a$ - končí na a
^a.*b$ - řádek začíná na a a končí na b
[a-z0-9] - libovolný znak z a-z nebo 0-9
[a-z]{3} - tři znaky z a-z
[a-z]{3,5} - tři až pět znaků z a-z
a|b - buď znak a nebo znak b
```

Sekvence znaků

- Sekvence znaků se používají pro hledání konkrétního řetězce znaků.
- Sekvence znaků se hledají přesně tak, jak jsou napsány.

```
1   abc - abc
2   abc{3} - abccc
3   abc{3,5} - abccc, abcccc, abccccc
```

Množiny znaků

```
[abc] - a nebo b nebo c
[a-z] - libovolný znak z a-z
[A-Z] - libovolný znak z A-Z
[0-9] - libovolný znak z 0-9
[a-zA-Z] - libovolný znak z a-z nebo A-Z
[a-zA-Z0-9] - libovolný znak z a-z, A-Z nebo 0-9
```

```
\d - libovolné číslo (0-9) [0-9]
\D - libovolný znak, který není číslo (0-9) [^0-9]
\w - libovolné písmeno nebo číslo (a-z, A-Z, 0-9)
\W - libovolný znak, který není písmeno nebo číslo (^a-z, A-Z, 0-9)
\s - libovolný bílý znak (mezera, tabulátor, nový řádek)
\S - libovolný znak, který není bílý znak
```

Skupiny znaků

- Skupiny znaků se používají pro hledání více znaků najednou.
- Skupiny se vždy uzavírají do závorek `()`.
- Skupiny mohou obsahovat libovolné znaky nebo zástupné znaky.

```
(a|b) - a nebo b
(a|b|c) - a nebo b nebo c
(a|b|c){3} - tři znaky a, b nebo c
(a|b|c){3,5} - tři až pět znaků a, b nebo c
([A-E]|[0-5]){1} - buď znak z A-E nebo z 0-5
([a-z]|[^0-9]){3} - tři znaky z a-z nebo znaky, které nejsou z 0-9
```

Negace

- Negace se používá pro hledání znaků, které nejsou v dané množině.
- Negace se zapisuje pomocí `^` na začátku množiny znaků.

```
[^a-z] - libovolný znak, který není z a-z
[^0-9] - libovolný znak, který není z 0-9
```

Pozor! Negace se používá pouze uvnitř množiny znaků `[]`. Neplést s začínáním řádku `^`.

Logický OR (Alternativa)

- Alternativa se používá pro hledání více možností najednou.
- Zapisujeme pomocí svislítka `|` mezi možnostmi.

```
a|b - buď znak a nebo znak b
a|A - buď znak a nebo znak A
abc|def - buď abc nebo def
[a-z]|[^0-9] - buď znak z a-z nebo znak, který není z 0-9
```

Regex prakticky

Postup při tvorbě regulárního výrazu

1. **Definujte, co hledáte.**
   - Např. "Hledám všechna slova, která začínají na `a` a končí na `z`."
2. **Rozdělte problém na části.**
   - Hledání slova: `[a-zA-Z]+`
   - Začíná na `a`: `^a`
   - Končí na `z`: `z$`
3. **Postupně testujte jednotlivé části.**
   - Začněte s jednoduchým výrazem a postupně přidávejte další pravidla.
4. **Použijte nástroje pro testování.**
   - Doporučené nástroje: Regex101 nebo RegExr.

Vizualizace regulárních výrazů

Např. pro regulární výraz `^a.*z$`:

1. `^`: Začátek řádku.
2. `a`: První znak musí být `a`.
3. `.*`: Jakékoliv znaky (nula nebo více opakování).
4. `z$`: Poslední znak musí být `z`.

::right::

Příklad

```
1  ^a.*z$
```

Text	Výsledek
abc	✗

Časté chyby při tvorbě regexů

1. **Zapomenutý escape znak**:
   - Např. hledáte tečku (`.`), ale bez escapování se jedná o zástupný znak.
   - Řešení: Použijte `\.`.
2. **Příliš obecný výraz**:
   - Např. `.*` může zachytit více, než potřebujete.
   - Řešení: Zúžit pomocí kvantifikátorů, např. `.{1,5}`.
3. **Nepoužité kotvy (`^` , `$`)**:
   - Pokud nehledáte od začátku nebo do konce řádku, může dojít k nechtěným výsledkům.
   - Řešení: Použit kotvy tam, kde jsou nutné.

Regex v Pythonu

Regulární výrazy v Pythonu

- Python má zabudovanou knihovnu `re` pro práci s regulárními výrazy.
- Knihovna obsahuje několik základních funkcí pro práci s regulárními výrazy.
- Regulární výrazy se vždy zapisují jako raw stringy (označené předponou `r`).
- `re._metoda_(_pattern_, _zpracovávaný text_)`
- Nejčastěji používané metody:
  - `re.search()`
  - `re.sub()`
  - `re.match()`
  - `re.findall()`

Příklady použití v Pythonu

```python
import re

# Hledání textu
re.search(r'abc', 'abcde') # <re.Match object; span=(0, 3), match='abc'>

# Nahrazení textu
re.sub(r'abc', 'def', 'abcde') # 'defde'

# Validace vstupu
re.match(r'[a-z]+', 'abc') # <re.Match object; span=(0, 3), match='abc'>

# Extrahování dat
re.findall(r'[a-z]+', 'abc def ghi') # ['abc', 'def', 'ghi']

# Filtraci dat
re.findall(r'[a-z]{3}', 'abc def ghi') # ['abc', 'def', 'ghi']
```

Příklad `re.sub()`

```python
import re

# Pro pattern vždy používejte raw stringy (označené předponou r),
# abyste se vyhnuli escapování zpětným lomítkem.

text = "Python je skvělý jazyk. Python je velmi populární."
pattern = r'Python'
replacement = "PHP"

new_text = re.sub(pattern, replacement, text)
print(new_text)  # "PHP je skvělý jazyk. PHP je velmi populární."
```

re.fullmatch()

```python
import re

# regularni vyraz
regex = r"[a-zA-Z]{2,10}"

text = "Jmeno"

if(re.fullmatch(regex, text)):
    print("Splňuje")
else:
    print("Nesplňuje")
```

re.search()

```python
import re

texty = ["ps","pes","pse","poe","prase","poklice"]
vyrazy = [ r"p[ars]e", r"p[ars]*e", r"p[^ars]e" ]

for text in texty:
    for vyraz in vyrazy:
        if re.search(vyraz, text):
            vysledek = "ano"
        else:
            vysledek = "ne"

        print(text + " splňuje " + vyraz + ": "+vysledek)
```

Word Boundaries

- Word boundaries se používají pro hledání slov.
- zapisují se pomocí `\b` na začátku nebo na konci slova.
- Když je umístěno mezi dvěma znaky, kontroluje, zda se nachází na hranici slova (tj. mezi slovem a ne-slovem).

```
\babc\b - abc
\babc\b - abc, abcde, abcdef, ...
```

```
# Najde slovo “python” pouze jako samostatné slovo,
# nikoli jako součást jiného slova (např. “pythonic”).
r"\bpython\b"

# Najde třímístná čísla jako samostatná slova (např. “123”, ale ne “1234”).
# Oba zápisy jsou ekvivalentní
r"\b\d{3}\b"
r"\b[0-9]{3}\b"
```

Děkuji za pozornost

Otázky?

Repository / Prezentace