

Wyrażenia regularne Java - klasa Matcher i Pattern

Beata Pańczyk

Wprowadzenie do wyrażeń regularnych

- **wyrażenia regularne** (ang. regular expressions, regexp lub regexes)- sposób opisywania tekstu poprzez dopasowywanie wzorców
- **zastosowanie** - bardziej złożone dopasowywanie ciągów (np. sprawdzanie poprawności dat, zastępowanie jednych fragmentów tekstu innymi, pobieranie fragmentów z większych bloków tekstu)
- 2 style składni wyrażeń regularnych: POSIX i Perl
- w połączeniu z konkretnym narzędziem, w którym zostały zaimplementowane, umożliwiają różnorodne sposoby przetwarzania tekstu.

2

Wyrażenia regularne - składnia

- **Znaki zwyczajne**
a, b, c, z, A, B, C, Z, 0, 1, 2, 9, ,, !, _, ...
- **Znaki specjalne (metaznaki)**
 - metaznaki rozpoznawane w dowolnym miejscu wzorca poza nawiasami kwadratowymi []
^, \$, ., *, ?, +, [,], {, }, (,), \
– metaznaki rozpoznawane w **klasach znaków** (część wzorca ujęta w nawiasy kwadratowe [], pasująca do dokładnie jednego znaku, bez względu na liczbę tworzących ją znaków)
^, -, \
]

3

Elementy wyrażeń regularnych: dopasowywanie tekstu

- Wszystkie znaki oprócz znaków specjalnych, określają same siebie, np:
k - określa łańcuch złożony ze znaku k
- Kolejne znaki oznaczają, że znaki te muszą wystąpić w łańcuchu dokładnie w takiej samej kolejności, np:
kot pozwala znaleźć łańcuch kot w dowolnym miejscu wiersza
RegExp oznacza RegExp

4

Elementy wyrażeń regularnych: znaki specjalne

- **Kropka .** - oznacza dowolny znak, z wyjątkiem znaku nowego wiersza, np:
 - r.k - pasuje do rok, rak, ryk itp.
 - .a - do mak, rak, lat itp.
 - .o.a - do lola, cola, wola, kolacja itp.
- **rozgałęzianie []** - oznacza „lub”, „OR” i pozwala na łączenie wielu wyrażeń w jedno, do którego pasuje dowolne z wyrażeń składowych np:
 - (gif)(jpg) - pasuje do gif lub jpg
 - (U|u)(l|lica) (3-go|Trzeciego) Maja - pasuje do ul 3-go Maja, Ul 3-go Maja, ulica 3-go Maja, Ulica 3-go Maja, ul Trzeciego Maja, Ul Trzeciego Maja, ulica Trzeciego Maja, Ulica Trzeciego Maja

5

Elementy wyrażeń regularnych: znaki specjalne - grupowanie

- Podwzorec może być zamknięty w niepodzielnej grupie za pomocą nawiasów (). W ten sposób można użyć gałęzi nie tylko dla całego wzorca, ale również dla jego fragmentów np:
 - Fizy(cy|k) - pasuje do Fizycy i Fizyk
- Zestaw znaków między nawiasami kwadratowymi oznacza dowolny znak objęty nawiasami kwadratowymi, np:
 - [1234], [1-4] - oznacza 1 lub 2 lub 3 lub 4
 - pi[wk]o - pasuje do piwo i piko
 - [a-z] - dopasowywane znaki ograniczamy do zbioru małych liter
 - [aeiouy] - wyluczanie elementów zbioru (samogłoski)
 - [a-zA-Z] - wszystkie małe i duże litery

6

Kotwiczenie

- **kotwiczenie:**
 - ^ - stosowany na początku wyrażenia regularnego w celu wskazania, że musi się ono pojawić na początku szukanego ciągu
 - \$ - stosowany na końcu wyrażenia regularnego, które musi się pojawić na końcu szukanego ciągu np.
 - kot - dopasuje łańcuch kot w dowolnym miejscu wiersza
 - ^kot - pasuje wtedy, gdy mamy początek wiersza, po którym od razu występuje litera k, po niej od razu litera o, a po niej od razu litera t
 - ^kot\$ - pasuje jeśli wiersz zawiera początek, po którym od razu znajdują się znaki kot, a po nich od razu koniec wiersza
 - ^pawel, gif\$, ^[a-z]\$ - (pasuje do każdego pojedynczego znaku a-z, jako osobnego ciągu)

7

Zakotwiczenia

- ^ (**daszek**) oznacza *nie*, kiedy jest umieszczony w nawiasach []
 - [^a-z] - każdy znak, który nie pochodzi z zakresu a-z
- Większość znaków specjalnych w tym miejscu traci swoje znaczenie, np:
 - [^piwo] - pasuje do wszystkich łańcuchów w których nie występuje słowo piwo
 - pi[^wk]o - pasuje np. do pinokio, ale wyklucza słowa: piwo oraz piko
- UWAGA - ze względu na to, że zarówno [-] jak i [^] mają specjalne znaczenie, jeśli umieścić je w nawiasach kwadratowych, dlatego też
 - chcąc dopasować daszek [^] nie należy umieszczać go na początku;
 - chcąc dopasować minus [-] najbezpieczniej umieścić go jako ostatni znak w zakresie;
 - dlatego zamiast [^%\$#@!], należy zastosować [%\$#@!^], a zamiast [a-c], chcąc dopasować 'a', 'c' lub '-' należy zastosować [ac-].

8

Reprezentacje znaków

- \a (alarm x07)
- \e (ESC x1B)
- \n (nowy wiersz x0A),
- \t (tab x09)
- \r (powrót karetki x0D)
- \f (wysunięcie kartki x0C)
- \x(szesnastkowa) - znak reprezentowany przez jedno- lub dwucyfrową liczbę szesnastkową
- \x{szesnastkowa} - znak reprezentowany przez dowolną liczbę szesnastkową
- **dopasowywanie specjalnych znaków literowych:**
 - \ - należy umieścić przed znakiem specjalnym
 - np. \\, \}, \\$, \.

9

Klasy znaków

- [...] - pojedynczy znak podany lub zawierający się w określonym zakresie
- [^...] - pojedynczy znak, który nie został podany lub nie zawiera się w określonym zakresie
- [[:klasa:]] - klasa znaków POSIX

10

Predefiniowane klasy znaków POSIX

- [[:alnum:]] - znaki alfanumeryczne
- [[:alpha:]] - znaki alfabetu
- [[:lower:]] - małe litery
- [[:upper:]] - duże litery
- [[:digit:]] - liczby dziesiętne
- [[:xdigit:]] - liczby szesnastkowe
- [[:punct:]] - znaki przestankowe
- [[:blank:]] - tabulatory i spacje
- [[:space:]] - pusta przestrzeń
- [[:cntrl:]] - znaki kontrolne
- [[:print:]] - wszystkie możliwe do wyświetlenia znaki
- [[:graph:]] - wszystkie możliwe do wyświetlenia znaki poza spacjami

11

Powtarzalność i podwyrażenia

- **powtarzalność** można określić stosując znaki specjalne:
 - * - wzór może powtórzyć się zero bądź więcej razy
 - + - wzór może powtórzyć się jeden bądź więcej razy
 - ? - wzór może wystąpić jeden bądź zero razy
 - np.
 - [[:alnum:]]+ - co najmniej jeden znak alfanumeryczny
 - ko?t pasuje do kt, kot
 - ko?t pasuje do kt, kot, koot, koooooot, ...
 - ko+t pasuje do kot, koot, koooooot, ...
- wyrażenie można rozdzielić na **podwyrażenia** stosując nawiasy jak w zwykłych wyrażeniach arytmetycznych np. (**bardzo**)**dużo** pasuje do 'dużo', 'bardzo dużo', 'bardzo bardzo dużo'

12

Powtarzalność

- **podwyrażenia policzalne** - ilość powtórzeń danego ciągu można określić stosując nawiasy klamrowe
- Wyrażenie {X} oznacza dokładnie X wystąpień
- Wyrażenie {X,} co najmniej X wystąpień, czyli przykładowo {0,} = *, {1,} = +
- Wyrażenie {,X} co najwyżej X wystąpień
- Wyrażenie {X,Y} oznacza Y dopasowań (jeśli to możliwe), ale do powodzenia wystarczy mu już X, np: {0,1} = ?
- np.:
 - {3} - dokładnie 3 powtórzenia
 - {2,4} - od dwu do czterech powtórzeń
 - {2,} - co najmniej dwa powtórzenia
 - np. **(bardzo){2,3}** - pasuje do 'bardzo bardzo', 'bardzo bardzo bardzo'

13

Znaki specjalne - zestawienie

- dopasowanie do każdego znaku oprócz nowej linii
 - (początek podciągu
 -) koniec podciągu
 - { początek minimalnego/maksymalnego kwantyfikatora
 - } koniec minimalnego/maksymalnego kwantyfikatora

W nawiasach kwadratowych wyrażen POSIX stosuje się:

- \ poprzedza znak specjalny (np. \\b, \\n)
- ^ NOT jeśli użyte przed wyrażeniem
- określenie zakresu znaków

14

Zastosowania

- Przetwarzanie tekstu:
 - walidacja formularzy
 - poprawianie pomyłek
 - masowa zmiana wyrazów w tekście
 - wyciąganie pewnych wyrazów pasujących do wzorca z tekstu
 - konwertowanie adresów www, generowanych dynamicznie na statyczne
 - i wiele innych operacji związanych z tekstem...

15

Zastosowanie wyrażeń regularnych

- sprawdzenie poprawności adresu pocztowego postaci: **user@serwer.domena** za pomocą wyrażenia regularnego: **^[a-zA-Z0-9_]+@[a-zA-Z0-9\.-]+\.[a-zA-Z0-9\.-]+\$**
- znaczenie podwyrażeń:
 - ^[a-zA-Z0-9_]+** początek ciągu to przynajmniej jedna litera, cyfra lub _ (albo kombinacja tych znaków)
 - @** znak @
 - [a-zA-Z0-9\.-]+** znaki alfanumeryczne i łączniki
 - \.** znak .
 - [a-zA-Z0-9\.-\+]+\$** litery, cyfry i łączniki oraz ewentualnie więcej kropek, i tak do końca ciągu

16

Walidacja

- Walidacja po stronie klienta i po stronie serwera
- Walidacja formularza po stronie klienta odbywa się w języku JavaScript i ma na celu jedynie wygodę użytkownika. Zabezpieczenia takie łatwo ominąć. Nie wolno ich traktować jako środka gwarantującego bezpieczeństwo serwisu.
- Wszelkie dane pochodzące od użytkownika, a zatem także dane przekazane za pomocą formularzy, należy traktować jako potencjalnie niebezpieczne. Pierwszym etapem działania skryptu powinna być walidacja wszystkich zmiennych.
- Walidacja danych w aplikacji internetowej powinna odbywać się w dwóch kierunkach: należy dokładnie sprawdzić zarówno wartości przesłane przez klienta do skryptu, jak i zabezpieczyć wyniki produkowane przez skrypt

17

Typ danych

- **Zmienne** przekazywane protokołem HTTP są napisami - żadna informacja dotycząca typu danej nie jest dołączana do zapytania HTTP - na najniższym poziomie, dane odebrane po stronie serwera są napisami.
- **Bez względu na rodzaj wprowadzonej informacji** (liczba całkowita, liczba rzeczywista, wartość logiczna, napis), w skrypcie **mamy do dyspozycji napis** odpowiedniej długości, który zawiera dane wprowadzone przez użytkownika bądź (jak to ma miejsce na przykład w przypadku kontrolki typu radio) wygenerowane przez przeglądarkę przed wysłaniem formularza

18

Walidacja napisów

- Długość napisu
- Dozwolone znaki
- Niedozwolone znaki
- Usuwanie kodu HTML
- Usuwanie białych znaków
- Standaryzacja danych: konwersja wielkości liter

19

Przykładowe wyrażenia regularne

- `www_reg = "^(((f|ht)tp?s?):V|www\\.)[/w-_-]+(\\.[w-_-]+)?([w-_-\\.@?^=%&v~!+#!]*[w-_-@?^=%&v~!+#!])?$";`
- `email_reg = "^([w-_-]+\\.?)@[/w-_-]?_?\\.?[a-z]{2,4}$";`

`http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html`

- `imie_reg = "[a-zA-Ząęłńóśźżąćęłńóśźż]{2,20}$";`
- `nazwisko_reg = "[a-zA-Ząęłńóśźżąćęłńóśźż]{2,40}$";`
- `login_reg = "[a-zA-Z0-9_-]{3,15}$";`
- `miasto_reg = "[a-zA-Ząęłńóśźżąćęłńóśźż]{2,50}$";`
- `tel_reg = "^[1-9]{1,1}[0-9]{1,1}(-)?[1-9]{1,1}[0-9]{6,6}([1-9]{1,1}[0-9]{8,8})$";`

20

Ciekawe linki

- Wyrażenia Regularne – Jeffrey E.F. Friedl
- <http://regexlib.com/>
- <http://www.rexx.org/>
- <http://osteele.com/tools/rework/>
- <http://www.programuj.com/artykuly/www/regularne.php?nolimit=true>
- http://maciek.lasyk.info/regexp_checker.html
- <http://grabun.com/wyrazenia-regularne/>
- <http://www.siteexperts.com/tips/functions/ts23/page1.asp>
- <http://www.sweeting.org/mark/html/revalid.php>
- <http://www.google.pl/>

21

Wyrażenia regularne w javie Klasa Pattern

- Do obsługi wyrażeń regularnych służą klasy **Matcher** i **Pattern** z pakietu **java.util.regex**
- Klasa **Pattern** nie ma konstruktorów a obiekt klasy zwracany jest przez statyczną metodę **compile()**, której argumentem jest ciąg znaków z wyrażeniem regularnym; **compile** powoduje transformację wyrażenia regularnego do wzorca, np.: **Pattern wzorzec = Pattern.compile("wyrażenie_regularne");**
- Metoda **matcher()** klasy **Pattern** zwraca obiekt klasy **Matcher**; argumentem metody jest ciąg znaków w którym będzie poszukiwany wzorzec, np.: **Matcher sekwencja=wzorzec.matcher("ciąg_do_przeszukania");**

22

Klasa Matcher

- Nie posiada konstruktorów a do tworzenia obiektu używa się metody **matcher()** klasy **Pattern**
- Metody:
 - **int end()** – zwraca indeks ostatniego znaku pasującej sekwencji powiększony o 1
 - **boolean find()** – zwraca true, jeśli znaleziono sekwencję pasującą do wzorca
 - **String group()** – zwraca ostatnio znalezioną pasującą sekwencję
 - **boolean matches()** – zwraca true jeśli sekwencja (lub jej fragment) pasuje do wzorca; wywołana po raz kolejny szuka kolejnego pasującego fragmentu
 - **String replaceAll(String nowy)** - przeszukuje ciąg znaków i zastępuje w nim pasujące sekwencje nowym ciągiem znaków
 - **int start()** - zwraca indeks pierwszego znaku pasującej sekwencji

23

Przykład 1

```
import java.util.regex.*;
public class WyrazeniaRegularne {
    public static void main(String[] args) {
        //utworzenie wzorca regex:
        Pattern wzorzec=Pattern.compile("[ks].+?e");
        //utworzenie sekwencji zawierającej wszystkie podciągi
        //pasujące do wzorca wyszukane w podanym ciągu:
        Matcher sekwencja=
            wzorzec.matcher("konie, żyrafy, stonie, psy i koty");
        //pobranie poszczególnych podciągów z sekwencji:
        while (sekwencja.find())
        { System.out.println(sekwencja.group()); //wynik: konie stonie
        }
    }
}
```

24

Przykład 2

Metoda pomocnicza:

```
public boolean pasuje(String input, String reg)
{ Pattern wzorzec=Pattern.compile(reg);
  Matcher sekwencja=wzorzec.matcher(input);
  return sekwencja.find();
}
```

Wywołanie w serwlecie:

```
String imie=request.getParameter("nazwisko");
imie = new String(imie.getBytes("ISO-8859-1"), "UTF-8");
String ereg="[a-zA-ZąćłńóśźżĄĆĘŁŃÓŚŻ]{2,20}$";
if ((imie!=null) && !(imie.trim().equals("")) && (pasuje(imie,ereg)) )
{ out.println("<p>Imię:"+imie+"</p>");}
else out.println("Błędne dane");
```

25