Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Reguläre Ausdrücke mit Java

Johannes Dyck, Thomas Schulz

Projektseminar www.ProminentPeople.info

Gliederung

1. Grundlagen von regulären Ausdrücken
    1. Aufbau
    2. Zeichenklassen
    3. Metazeichen
    4. Quantoren
    5. Beispiele
2. Reguläre Ausdrücke in Java
    1. Replace
    2. Pattern
    3. Matcher
3. Anwendungen in ProminentPeople

Grundlagen von regulären Ausdrücken

- Zeichenkette, zur Beschreibung von Mengen von Zeichenketten
- theoretische Informatik – reguläre Sprachen, endliche Automaten
- *komplizierte* Textersetzungen (Mustererkennung)
- Erzeugung von Wörtern (*Schablone*)

Aufbau von regulären Ausdrücken

- bestehen aus Literalen, Meta-Zeichen & Zeichenklassen
- in Java wie Strings dargestellt
- „abv“ passt auf gfjsdp***abv***ajfsgofdsklös nicht auf abw

Metazeichen I

- **. | ^ $ \ [ ] - ( ) { }** und Quantoren

- **.** beliebiges Zeichen (außer Zeilenumbruch)
- **|** Alternativen
- **^** Negation bei Zeichenklassen
  - (Zeilenanfang)
- **$** (Zeilenende)
- **** *Escape*-Operator

Metazeichen II

- **. | ^ $ \ [ ] - ( ) { }** und Quantoren

- **[]** Definition von Zeichenklassen
- **-** notwendig zur Verwendung von Zeichenklassen
- **()** Gruppierung
- **{}** Mengenangaben
  - {min,max} {0,3} {5,} {,10} {2}

Metazeichen III (Quantoren I)

- `. | ^ $ \[ ] - ( ) { }` und **Quantoren**

- **?** Optionalität {0,1}
- **+** Existenzquantor {1, ∞}
- ***** Allquantor {0, ∞}

Quantoren II

- *gieriges* Verhalten: Standard
  - findet größtmögliche Zeichenkette
  - Bsp.: .*foo findet in xfooxxxfoo xfooxxxfoo
- *gemäßigt*: Anstellen von ?
  - bricht früher ab als
  - Bsp.: .*?foo findet in xfooxxxfoo xfooxxxfoo und xfooxxxfoo
- *possesiv*: Anstellen von +
  - versucht nur einmal größtmöglichen Treffer zu finden
  - Bsp.: .*+foo findet nichts in xfooxxxfoo

Zeichenklassen

- beschreiben von Mengen von Zeichen
- vordefinierte
  - Ziffer **\d**
  - keine Ziffer **\D**
  - Buchstabe, Ziffer, Unterstrich **\w**
  - kein Buchstabe **\W**
  - „Whitespaces“ **\s** (Leerzeichen, Zeilenumbrüche, Tabs)
  - nicht-Whitespaces **\S**
- selbstdefinierte
  - [a-g] = [abcdefg]
  - [0-9a-zA-z_,;\.\(\)-]
  - [^äöüßÄÖÜ]

Randfunde („Boundary Matchers“)

- ^ Zeilenanfang
- $ Zeilenende
- \b Wortgrenze
  - "\b\w+\b" = "\s\w+\s"
- \B Nicht-Wortgrenze
  - "\B\w+\B" = "\S\w+\S"
- \G Ende vom vorherigen Fund
  - \Gxyz != xyz
  - xyz xyz

Beispiele

- Name:

  ([A-ZÄÖÜ]{1}(\.|[a-zäöüß]+))?(\s[A-ZÄÖÜ]{1}[a-zäöüß]+)+

  - davor: Titel
  - dazwischen: Zwischennamen (von, van, …)
- eMail-Adresse: [a-z_\-\.]+@[a-z_\-\.]+\.[a-z]{2,4}
- Straße:

  ([A-ZÄÖÜ]{1}[a-zäöüß]*(\-| |\.))+[1-9]?[\d{0,4}]([a-z]|[A-Z])?
- Geburtsdatum: \d{1,2}\.\d{1,2}\.(\d{2}|\d{4})
  - ((0[1-9]|[1-2][0-9])\.){2}(19|20)\d{2}
- PLZ: ^([A-Z]{1,3}-)?\d{5}$

Reguläre Ausdrücke mit Java

- `public String replaceAll`

  `(String regex, String replacement)`
  - Ersetzen aller gefundenen Vorkommen des regulären Ausdruckes regex durch replacement
  - folgt den normalen Regeln für reguläre Ausdrücke
  - `String s = "aaa";`
  - `s.replaceAll("a*?", "b");`
  - `S = "bababab";`

- `Public String replaceFirst(..)`
  - analog

java.util.regex.Pattern

- regulärer Ausdruck als String gegeben
- muss kompiliert werden, um angewendet zu werden
  - `Public static Pattern Pattern.compile (String regex)`
- Objekt der Klasse Pattern wird erzeugt
- public Matcher matcher(String s)
  - Anwendung des kompilierten Audrucks auf s

java.util.regex.Matcher

- erzeugt durch Methode matcher() in Klasse Pattern
- `String group(int i)`
  - gibt String zurück, auf den das i-te Klammerpaar des Ausdrucks zutrifft
  - bei Index 0 wird der gesamte Ausdruck betrachtet
- `int start(group int) / int end(group int)`
  - Index bei Anfang/nach der angegebenen Gruppe

Anwendungen für www.prominentpeople.info

- `s.replaceAll("&auml;", "ä");`
- `s.replaceAll("<!--.+?-->", " ");`
- `s.replaceAll(">([a-zA-ZÄÖOäöüß])+?<", " ");`
- `s.replaceAll("<.+?>", " ");`
  - `s.replaceAll("<.+>", " ");`

Anwendungen für www.prominentpeople.info

- `p = Pattern.compile("<.?body.+?>(.+?)</.?body>");`
  - Erkennt durch body und ähnliche Tags eingeschlossene Strukturen …
  - … da in den Sourcen kein einheitliches body-Tag verwendet wird
- `String body = m.group(1);`
- `array = body.split("[^a-zA-ZäöüÄÖÜß]");`

Quellen

- http://java.sun.com/docs/books/tutorial/essential/regex
- http://de.wikipedia.org/wiki/Regulärer_Ausdruck
- Reguläre Ausdrücke, O'Reilly Verlag, Jeffrey E. F. Friedl, 2. deutsche Auflage 2004

Danke für die Aufmerksamkeit