Einführung in Reguläre Ausdrücke

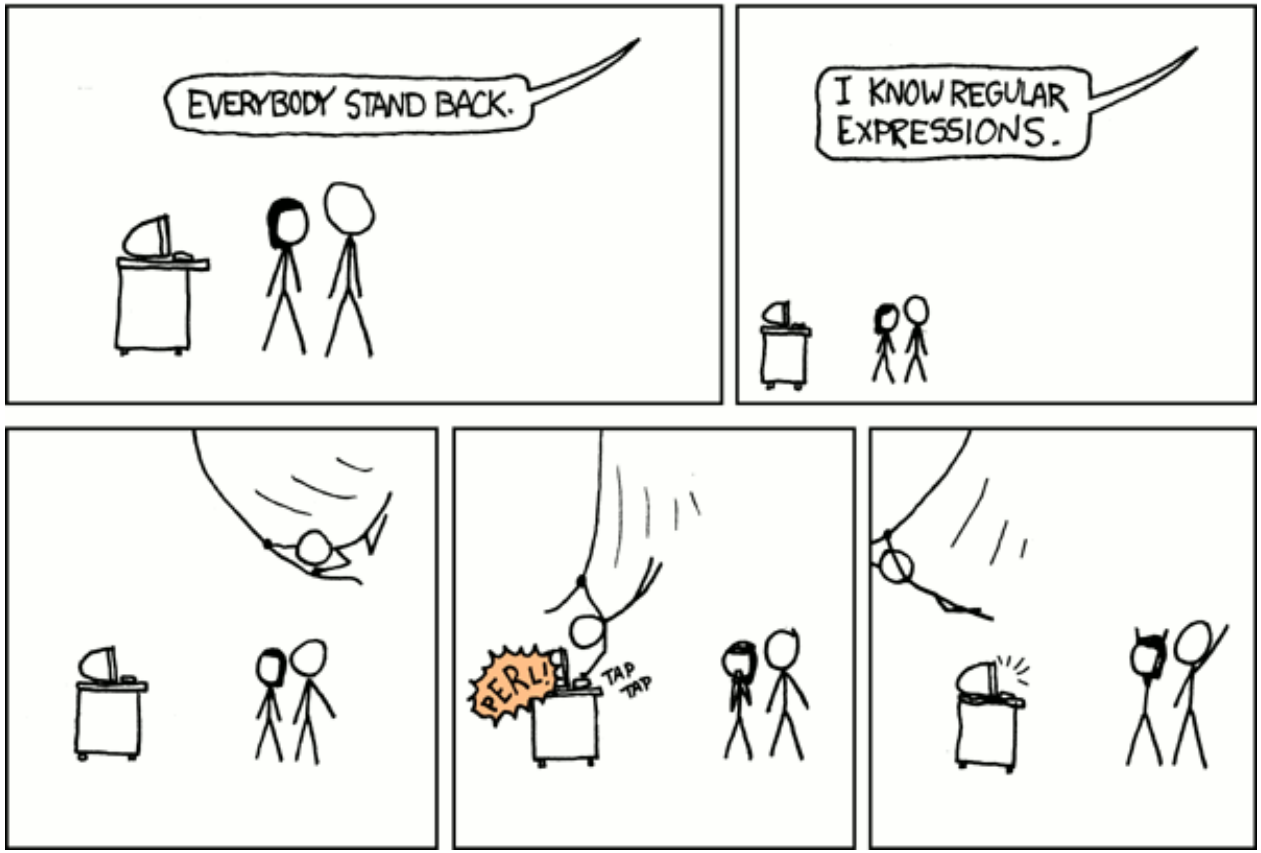
Sven Übelacker, Klaus Vormweg

18.09.2010 / Software Freedom Day

xkcd


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!

Fortsetzung ...



Grundlegendes

Begriff:

Reguläre Ausdrücke = regular expressions
kurz: regex

Funktion:

Reguläre Ausdrücke definieren flexible Muster, mit denen in Texten gesucht (und ersetzt) werden kann. Diese Muster können auch zur Eingabekontrolle verwandt werden.

Grundlegendes

Begriff:

Reguläre Ausdrücke = regular expressions
kurz: regex

Funktion:

Reguläre Ausdrücke definieren flexible Muster, mit denen in Texten gesucht (und ersetzt) werden kann. Diese Muster können auch zur Eingabekontrolle verwandt werden.

Theorie

- Reguläre Sprache -> Computerlinguistik/Automatentheorie
- eigentlich Typ 3 Grammatik in der Chomsky Hierarchie durch backreference erweitert
- POSIX Regexp ähnelt einer Regulären Sprache
- Darstellung meist in Extended Backus-Naur Form

Theorie

- Reguläre Sprache -> Computerlinguistik/Automatentheorie
- eigentlich Typ 3 Grammatik in der Chomsky Hierarchie durch backreference erweitert
- POSIX Regexp ähnelt einer Regulären Sprache
- Darstellung meist in Extended Backus-Naur Form

Theorie

- Reguläre Sprache -> Computerlinguistik/Automatentheorie
- eigentlich Typ 3 Grammatik in der Chomsky Hierarchie durch backreference erweitert
- POSIX Regexp ähnelt einer Regulären Sprache
- Darstellung meist in Extended Backus-Naur Form

Theorie

- Reguläre Sprache -> Computerlinguistik/Automatentheorie
- eigentlich Typ 3 Grammatik in der Chomsky Hierarchie durch backreference erweitert
- POSIX Regexp ähnelt einer Regulären Sprache
- Darstellung meist in Extended Backus-Naur Form

Geschichte

1968 Ken Thompson's qed (erster Editor mit Regex-Unterstützung)

1970er ed, grep, egrep

1986 Henry Spencer's Open Source Regex-Library

1987 Perl 1.1

1997 Perl Compatible Regular Expression Library (PCRE) (zunächst für Exim)

Geschichte

1968 Ken Thompson's qed (erster Editor mit Regex-Unterstützung)

1970er ed, grep, egrep

1986 Henry Spencer's Open Source Regex-Library

1987 Perl 1.1

1997 Perl Compatible Regular Expression Library (PCRE) (zunächst für Exim)

Geschichte

1968	Ken Thompson's qed (erster Editor mit Regex-Unterstützung)
1970er	ed, grep, egrep
1986	Henry Spencer's Open Source Regex-Library
1987	Perl 1.1
1997	Perl Compatible Regular Expression Library (PCRE) (zunächst für Exim)

Geschichte

1968	Ken Thompson's qed (erster Editor mit Regex-Unterstützung)
1970er	ed, grep, egrep
1986	Henry Spencer's Open Source Regex-Library
1987	Perl 1.1
1997	Perl Compatible Regular Expression Library (PCRE) (zunächst für Exim)

Geschichte

1968	Ken Thompson's qed (erster Editor mit Regex-Unterstützung)
1970er	ed, grep, egrep
1986	Henry Spencer's Open Source Regex-Library
1987	Perl 1.1
1997	Perl Compatible Regular Expression Library (PCRE) (zunächst für Exim)

Definitionen 1

Grundregeln:

Die “Sprache“ der Reguläre Ausdrücke besteht im wesentlichen aus Zeichen: Es gibt Metazeichen, die Funktionen haben und literale Zeichen, die für sich stehen (Terminale). Um Metazeichen als Literale zu behandeln, werden sie mit dem Backslash “\“ entwertet.

Fakten:

Reguläre Ausdrücke gibt es in verschiedenen “Geschmacksrichtungen“: z.B. *POSIX-konform* in basic und extended (X), *GNU* (G), *Perl* (P) und viele mehr.

Definitionen 1

Grundregeln:

Die “Sprache“ der Reguläre Ausdrücke besteht im wesentlichen aus Zeichen: Es gibt Metazeichen, die Funktionen haben und literale Zeichen, die für sich stehen (Terminale). Um Metazeichen als Literale zu behandeln, werden sie mit dem Backslash “\“ entwertet.

Fakten:

Reguläre Ausdrücke gibt es in verschiedenen “Geschmacksrichtungen“: z.B. *POSIX-konform* in basic und extended (X), *GNU* (G), *Perl* (P) und viele mehr.

Definitionen 2

Wie funktionieren Regexes:

- Reguläre Ausdrücke werden zeichenweise von links nach rechts abgearbeitet
- Alle Zeichen ohne besondere Bedeutung passen auf genau ein Zeichen im Vergleichstring
- Dieses einfache Match-Verhalten kann mit Quantoren, Wildcards etc. beinflusst werden
- Reguläre Ausdrücke sind standardmäßig “gefräßig“, d.h. sie versuchen, soviel Text wie möglich zu matchen

Definitionen 2

Wie funktionieren Regexes:

- Reguläre Ausdrücke werden zeichenweise von links nach rechts abgearbeitet
- Alle Zeichen ohne besondere Bedeutung passen auf genau ein Zeichen im Vergleichstring
- Dieses einfache Match-Verhalten kann mit Quantoren, Wildcards etc. beinflusst werden
- Reguläre Ausdrücke sind standardmäßig “gefräßig“, d.h. sie versuchen, soviel Text wie möglich zu matchen

Definitionen 2

Wie funktionieren Regexes:

- Reguläre Ausdrücke werden zeichenweise von links nach rechts abgearbeitet
- Alle Zeichen ohne besondere Bedeutung passen auf genau ein Zeichen im Vergleichstring
- Dieses einfache Match-Verhalten kann mit Quantoren, Wildcards etc. beinflusst werden
- Reguläre Ausdrücke sind standardmäßig “gefräßig“, d.h. sie versuchen, soviel Text wie möglich zu matchen

Definitionen 2

Wie funktionieren Regexes:

- Reguläre Ausdrücke werden zeichenweise von links nach rechts abgearbeitet
- Alle Zeichen ohne besondere Bedeutung passen auf genau ein Zeichen im Vergleichstring
- Dieses einfache Match-Verhalten kann mit Quantoren, Wildcards etc. beinflusst werden
- Reguläre Ausdrücke sind standardmäßig “gefräßig“, d.h. sie versuchen, soviel Text wie möglich zu matchen

Definitionen 2

Wie funktionieren Regexes:

- Reguläre Ausdrücke werden zeichenweise von links nach rechts abgearbeitet
- Alle Zeichen ohne besondere Bedeutung passen auf genau ein Zeichen im Vergleichstring
- Dieses einfache Match-Verhalten kann mit Quantoren, Wildcards etc. beinflusst werden
- Reguläre Ausdrücke sind standardmäßig “gefräßig“, d.h. sie versuchen, soviel Text wie möglich zu matchen

Wo begegnen uns Reguläre Ausdrücke?

Editoren:

Viele Editoren bieten im Suchen/Ersetzen-Dialog die Möglichkeit, Reguläre Ausdrücke einzusetzen, z.B. *MS Word*, *OpenOffice*, *vi/vim* ...

Shell-Tools:

sed, *awk*, *grep/egrep* ...

SQL:

Oracle, *MySQL* ...

Wo begegnen uns Reguläre Ausdrücke?

Editoren:

Viele Editoren bieten im Suchen/Ersetzen-Dialog die Möglichkeit, Reguläre Ausdrücke einzusetzen, z.B. *MS Word*, *OpenOffice*, *vi/vim* ...

Shell-Tools:

sed, *awk*, *grep/egrep* ...

SQL:

Oracle, *MySQL* ...

Wo begegnen uns Reguläre Ausdrücke?

Editoren:

Viele Editoren bieten im Suchen/Ersetzen-Dialog die Möglichkeit, Reguläre Ausdrücke einzusetzen, z.B. *MS Word*, *OpenOffice*, *vi/vim* ...

Shell-Tools:

sed, *awk*, *grep/egrep* ...

SQL:

Oracle, *MySQL* ...

Wo begegnen uns Reguläre Ausdrücke 2?

Programmiersprachen:

Viele Programmiersprachen implementieren Reguläre Ausdrücke: *Perl*, *Java*, *JavaScript*, *Python*, *C*, *C++*, *Ruby*, *PHP*, *.NET*, *TCL*, *Haskell* ...

Beispiele für reguläre Ausdrücke

Zerlegt einen Text in Teile

```
^(([^:()]+):␣+)?([^()]+)(␣+(\([^()0-9]+\)))?
+\(([^()]+)\)\.␣+(.+)$
```

Findet einen Hyperlink im Text

```
https?://[a-z0-9-]+(\.[a-z0-9-]+)*\.
[a-z]{2,6}[-a-z0-9_:@&?=+,.!/~%$]*
(?>![.,?!:])
```

Anker

Metazeichen:

^: Anfang der Zeile
$: Ende der Zeile

Beispiele:

^#: Alle Zeilen eines Shellskripts, die auskommentiert sind
^$: Alle leeren Zeilen einer Datei

Zeichenklassen

Metazeichen:

`.`: Ein beliebiges Zeichen
`[]`: Eine Zeichenklasse `[aeiou]`

Metazeichen in Zeichenklassen:

`-`: definiert Zeichenbereich (nicht am Anfang oder Ende): `[a-z]`
`^`: Verneinung (nur am Anfang) `[^aeiou]`

Beispiele:

`<[hH][1-6]>`: findet alle öffnenden HTML-Überschrifttags
`^[^#]`: Alle Zeilen eines Shellskripts, die nicht auskommentiert sind

Alternativen und Gruppierung

Metazeichen:

`|` : Alternative
`(␣)` : Gruppierung

Beispiele:

`Juni|Juli` oder auch `Ju(n|l)i` - alternativ zu `Ju[nl]i`
Die Alternative “bindet“ immer bis zum nächsten Leerzeichen, sonst muss man Gruppieren:
`geehrte(r␣Herr|␣Frau)`

Quantoren 1: Optionales

Metazeichen:

`?`: Das Zeichen oder die Gruppe links vom Fragezeichen darf vorkommen oder auch nicht

Beispiel:

`</?[hH][1-6]>` findet öffnende und schließende Tags

Quantoren 2: “Gefräßig“

Metazeichen:

`*`: beliebig oft, auch gar nicht
`+`: beliebig oft, mindestens einmal
`{count}`: genau *count* mal
`{min,max}`: mindestens *min* mal, höchstens *max* mal

Beispiele:

`<p␣+class="test"␣*>`: Findet alle Vorkommnisse von Absätzen der Klasse “test“
`\.[a-z]{2,6}$`: Findet Domainendungen (.com, .de etc.) am Zeilenende

Quantoren 3: “Nicht Gefräßig“

Metazeichen:

`*?`: beliebig oft, auch gar nicht (P)
`+?`: beliebig oft, mindestens einmal (P)

Beispiele:

`<b>.*</b>`: Matcht auf
a rose is a rose is a rose
`<b>.*?</b>`: Matcht auf
a rose is a rose is a rose

Wortgrenzen

Metazeichen:

`\b`: Wortgrenze (G)(P)
`\B`: *keine* Wortgrenze (G)(P)

Beispiele:

`\bTest\b`: findet das Wort “Test“, aber nur als ganzes Wort.
Wortgrenzen finden kein Zeichen, sondern einen Ort in der zu prüfenden Zeichenkette

Backreferences

Metazeichen:

`(  )`: speichert Wert
`\1`: enthält Wert der ersten Klammer
`(?:  )`: gruppiert, speichert aber keinen Wert (P)

Beispiele:

`([a-zA-Z-])+␣\1` findet doppelte Vorkommen von Worten hintereinander in einem Text

Perl-Zeichenklassen

Metazeichen:

`\w`: “Wort“-Zeichen (Buchstaben und Unterstrich) (G) (P)
`\s`: Leerzeichen (+Tabulator) (G) (P)
`\d`: Ziffern (P)

Erläuterung:

Wird die Zeichenklasse groß geschrieben, bedeutet das Verneinung: `\D` sind alle Zeichen, die keine Ziffern sind.

Beispiele - grep

grep

```
grep -ve '^#'
grep -ve '^\s*#'
grep -ve '^\s*$'
```

Beispiele - Bash1

Zufallszahlen

```
# $rand mit 4 Zufallszahlen füllen:
[[ "$RANDOM$RANDOM$RANDOM$RANDOM"
   =~ ([0-9]{4}) ]] &&
rand="${BASH_REMATCH[1]}"
```

Beispiele - Bash2

IP-Adressen

```
# wenn $line eine IPv4 Adresse enthält,
# die erste in $ip schreiben:
if [[ "${line}" =~ ([0-9]{1,3}\.[0-9]{1,3}
      \.[0-9]{1,3}\.[0-9]{1,3}) ]]
then ip="${BASH_REMATCH[1]}"
fi
```

Beispiele - E-Mail

E-Mail-Adressen überprüfen

Praktikable Lösung für PCRE:

```
/\b([a-z0-9][a-z0-9._-]*@[a-z0-9-]+
(\.[a-z0-9-]+)*\.[a-z]{2,6})\b/i
```

Weiterführende Informationen

Online:

- Wikipedia: http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck
- Cheat Sheet von Dave Child: http://www.addedbytes.com/cheat-sheets/regular-expressions-cheat-sheet/
- perldoc perlre (http://perldoc.perl.org/perlre.html) oder kürzer: perldoc perlrequick (http://perldoc.perl.org/perlrequick.html)
- perldoc perlretut (Tutor)
- http://www.regular-expressions.info

Weiterführende Informationen

Buch:

Jeffrey Friedl: Mastering Regular Expressions (3. Auflage), O'Reilly 2008

Fragen?

Quellennachweis:

- xkcd comic strip „Regular Expressions“ unter CC by-nc 2.5

Download:

- http://www.tu-harburg.de/~psvkv/regex/regexec3.pdf

Fragen?

Quellennachweis:

- xkcd comic strip „Regular Expressions“ unter CC by-nc 2.5

Download:

- http://www.tu-harburg.de/~psvkv/regex/regexec3.pdf