

Die wunderbare Welt der Regulären Ausdrücke

Jakob Westhoff

International PHP Conference
11.10.10

Über mich

Jakob Westhoff



- Professioneller PHP Entwickler seit mehr als 9 Jahren
- Trainer und Consultant
- Autor von Artikeln und einem Buch
- Aktiv in verschiedenen Open-Source Projekten (arbit, PPC, phUML)
- Hauptanteilmäßig tätig für die crosscan GmbH

Fragen?

Fragen!

Terminologie

Terminologie



RegExp



Subject



Match

Terminologie



RegExp

○ Regulärer Ausdruck

- Beschreibung einer Menge von Zeichen und Zeichenketten

Terminologie



Subject

Satzgegenstand

- Zeichenkette auf die eine RegExp angewendet wird

Terminologie



Match

Treffer

- Der Teil der Zeichenkette, der Bestandteil der RegExp ist

RegExp

Aufbau einer RegExp

`/foobar/i`

Aufbau einer RegExp

`/foobar/i`



○ Delimiter

- Trennung zwischen Pattern und Modifiern

Aufbau einer RegExp

(foobar) i



The diagram shows the regular expression `(foobar) i` in red text. Two black arrows point upwards from below the text to the opening parenthesis `(` and the closing parenthesis `)`, highlighting their role as delimiters.

● Delimiter

- In PCRE sind Klammern erlaubt
 - () [] { }

Aufbau einer RegExp

(foobar) i



Pattern

- Beschreibung der gesuchten Zeichenkette

Aufbau einer RegExp

(foobar) i



Modifier

- Zusätzliche Optionen

Einfach eine Zeichenkette

- Das Pattern einer RegExp ist ein String

Open Source

Einfach eine Zeichenkette

- Das Pattern einer RegExp ist ein String

(Open Source)

Einfach eine Zeichenkette

- Das Pattern einer RegExp ist ein String

(Open Source)

Free and Open Source Conference

Einfach nur ein String

- Das Pattern einer RegExp ist ein String

(Open Source)

Free and Open Source Conference

- Das Pattern muss mind. einmal auftauchen
- Die Position innerhalb des Subject ist egal

Metacharacters

- Einige Zeichen innerhalb einer RegExp besitzen eine besondere Bedeutung

([0p]en \s* So+u.ce)

Quantifier

Quantifier

- Quantifier definieren Wiederholungen des vorherigen Zeichens oder Gruppe

(Op*en So+ur?c{1,3}e)

Quantifier

- Quantifier definieren Wiederholungen des vorherigen Zeichens oder Gruppe

(0p*en So+ur?c{1,3}e)



- * Beliebig oft ($0 \rightarrow \infty$)

Quantifier

- Quantifier definieren Wiederholungen des vorherigen Zeichens oder Gruppe

(Op*en So+ur?c{1,3}e)



- * Beliebig oft ($0 \rightarrow \infty$)
- + Beliebig oft aber mind. einmal ($1 \rightarrow \infty$)

Quantifier

- Quantifier definieren Wiederholungen des vorherigen Zeichens oder Gruppe

$(Op^*en\ So+ur?c\{1,3\}e)$



- $*$ Beliebig oft ($0 \rightarrow \infty$)
- $+$ Beliebig oft aber mind. einmal ($1 \rightarrow \infty$)
- $?$ Null oder einmal ($0 \rightarrow 1$)

Quantifier

- Quantifier definieren Wiederholungen des vorherigen Zeichens oder Gruppe

(Op*en So+ur?c{1,3}e)



- * Beliebig oft ($0 \rightarrow \infty$)
- + Beliebig oft aber mind. einmal ($1 \rightarrow \infty$)
- ? Null oder einmal ($0 \rightarrow 1$)
- {x,y} Zwischen x und y mal ($x \rightarrow y$)

Der Punkt

Der Punkt

- Der Punkt (.) matcht jedes beliebige Zeichen
 - außer einem Zeilenvorschub

(Ohne Punkt und .omma)



Der Punkt

- Der Punkt (.) matcht jedes beliebige Zeichen
 - außer einem Zeilenvorschub

(Ohne Punkt und .omma)

Ohne Punkt und Komma ✓

Der Punkt

- Der Punkt (.) matcht jedes beliebige Zeichen
 - außer einem Zeilenvorschub

(Ohne Punkt und .omma)

Ohne Punkt und Komma ✓

Ohne Punkt und Somma ✓

Der Punkt

- Der Punkt (.) matcht jedes beliebige Zeichen
 - außer einem Zeilenvorschub

(Ohne Punkt und .omma)

Ohne Punkt und Komma ✓

Ohne Punkt und Somma ✓

Ohne Punkt und _omma ✓

Der Punkt

- Umschalten der Engine auf single line

- Modifier **s**

(Ein .unkt)s



- Der Punkt matcht ebenfalls Zeilenvorschub

Zeichenklassen

Zeichenklassen

- Zeichenklassen definieren eine Menge beliebiger Zeichen

a b c d e f

Zeichenklassen

a b c d e f

Zeichenklassen

abcdef

- Keinerlei Trennzeichen

Zeichenklassen

[abcdef]

- Keinerlei Trennzeichen
- Umschlossen von eckigen Klammern ([])

Zeichenklassen

([abcdef] +)

- Keinerlei Trennzeichen
- Umschlossen von eckigen Klammern ([])
- Werden behandelt wie ein Zeichen

Zeichenklassen

$([a-f]^+)$

- Können als Bereiche definiert werden

Zeichenklassen

$([a - cd - f]^+)$

- Können als Bereiche definiert werden
- Dürfen aus mehreren Bereichen bestehen

Zeichenklassen

([abc.] +)

- Sonderzeichen verlieren ihre Bedeutung

Zeichenklassen

([abc . -]+)

- Sonderzeichen verlieren ihre Bedeutung
- Neue Sonderzeichen existieren

Zeichenklassen

([^]abcdef⁺)

- Negationen sind möglich

Zeichenklassen

([^]abcdef⁺)

- Negationen sind möglich
- Zeilenvorschub ebenfalls in der Negation

Zeichenklassen

`([^\n]+)`

- Negationen sind möglich
- Zeilenvorschub ebenfalls in der Negation
- Zeilenvorschub kann ausgeschlossen werden

Zeichenklassen

- Es gibt vordefinierte Zeichenklassen
 - `\d` Jede Ziffer (0,1,2,...)
 - `\s` Jeder Whitespace (<Space>, <Tab>, ...)
 - ...
- Großbuchstaben negieren die Klasse
 - `\D` Alles außer Ziffern
 - ...

Escaping

Escaping

- Besondere Bedeutung von Zeichen kann abgeschaltet werden (Escaping)

`jakob.westhoff@gmail.com`

Escaping

- Besondere Bedeutung von Zeichen kann abgeschaltet werden (Escaping)

`(jakob.westhoff@gmail.com)i`

Escaping

- Besondere Bedeutung von Zeichen kann abgeschaltet werden (Escaping)

(jakob.westhoff@gmail.com)i



- Dies ist ein echter Punkt und nicht ein beliebiges Zeichen

Escaping

- Besondere Bedeutung von Zeichen kann abgeschaltet werden (Escaping)

(jakob\.westhoff@gmail\.com)i



- Einsatz des Backslashes (\)

Escaping

- Funktioniert für alle Zeichen mit besonderer Bedeutung

(\ [\]) i

Escaping


- Funktioniert für alle Zeichen mit besonderer Bedeutung

`(*)i`
↑

Escaping

- Funktioniert für alle Zeichen mit besonderer Bedeutung

`(\ (\)) i`



Escaping

- Funktioniert für alle Zeichen mit besonderer Bedeutung

`(\+)i`



Escaping

- Funktioniert für alle Zeichen mit besonderer Bedeutung



Escaping in der Realität

- RegExp sind meist ein String

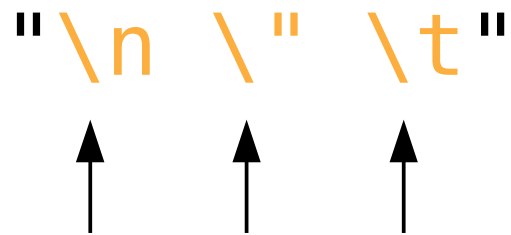
"(jakob\\.westhoff@gmail\\.com)i"



Escaping in der Realität

- Strings besitzen oft eigenes Escaping

"\n \" \t"



The diagram shows a string enclosed in double quotes: `"\n \" \t"`. The characters `\n`, `\`, and `\t` are highlighted in orange. Three black arrows point upwards from below to the backslash character in each of these three escaped sequences.

Escaping in der Realität

- Backslashes (\) in einem RegExp-String müssen escaped werden

"(jakob\\.westhoff@gmail\\.com)i"



Anker

Anker

- Anker gehören zur Familie der Assertions
 - to assert sth. = etw. durchsetzen
- Durchsetzung von Bedingungen unabhängig vom Match
- Hier: Anfang und Ende eines Subjects

Anker

- ⦿ ^ Anfang des Subject
- ⦿ \$ Ende des Subject

Anker

- ⦿ ^ Anfang des Subject
- ⦿ \$ Ende des Subject

(Ana) i

Anker

- ⦿ ^ Anfang des Subject
- ⦿ \$ Ende des Subject

(Ana) i

Ananas

Anker

- ^ Anfang des Subject
- \$ Ende des Subject

(Ana) i

Ananas ✓

Anker

- ⊙ ^ Anfang des Subject
- ⊙ \$ Ende des Subject

(Ana) i

Ananas ✓

Banane ✓

Anker

- ^ Anfang des Subject
- \$ Ende des Subject



(^Ana)i

Ananas ✓

Banane ✗

Anker (Modifier)

- Multiline Mode

- Modifier **m**

(**^abcdef\$**)**m**



- Anker treffen Anfang und Ende jeder Zeile innerhalb des Subject

Anker (Modifier)

(^abcdef\$)

abcdef
ghijkl
mnopqr

Anker (Modifier)

(`^abcdef$`)

abcdef
ghijkl
mnopqr

- Kein Match, da Anker Anfang und Ende des Subjekt Strings sind

Anker (Modifier)

$(^abcdef\$)m$



abcdef
ghijkl
mnopqr

- Anker sind nun Anfang und Ende jeder Zeile

Anker

- Newline Mode (**m**) unabhängige Anker
- **\A** Anfang des Subject
- **\z** Ende des Subject

Subpattern

Subpattern

- Runde Klammern unterteilen Pattern

`((abc)(def))`

`abcdef`

Subpattern

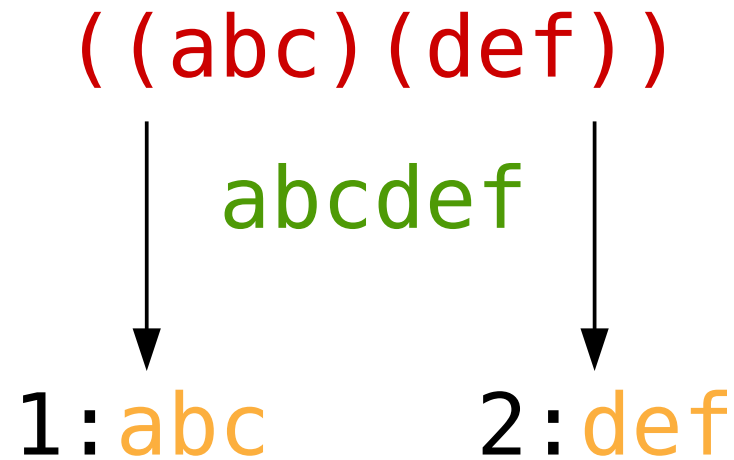
- Runde Klammern unterteilen Pattern

`((abc)(def))`

`abcdef`

Subpattern

- Runde Klammern unterteilen Pattern



- Extraktion von Teilen eines Matches

Subpattern

`((abc)\1)`

`abcabc`

- Wiederverwendung innerhalb eines Pattern

Benannte Subpattern

- Subpattern können benannt werden

`((?P<Vorname>Jakob))`



- Einsatz der Option **P**

Benannte Subpattern

`((?P<Vorname>Jakob))`

Jakob Westhoff

Benannte Subpattern

`((?P<Vorname>Jakob))`



Jakob Westhoff

Vorname: Jakob

- Extraktion mit Benennung

Subpattern ohne Treffer

- Subpattern müssen keinen Teiltreffer erzeugen

((?:Jakob))



- Der Doppelpunkt im Subpattern

Alternativen

Alternativen

(Open Source | Linux)

Open Source

Linux

Open Source Linux

Lesbarkeit

Lesbare RegExp

- Kommentare, Einrückungen und Zeilenumbrüche in RegExp
 - Modifizier **x**

(foobar) x



Lesbare RegExp

```
(^[a-z0-9_%. - ]+@[a-z0-9. - ]+\.[a-z]{2,4}$)iD
```

Lesbar? Verständlich? Wartbar?

Lesbare RegExp

```
(  
  ^           #Start des Subject  
  [a-z0-9_%. - ]+ #Benutzer  
  @           #Trennzeichen @  
  [a-z0-9. - ]+ #Domain  
  \.         #Trennzeichen .  
  [a-z]{2,4} #TLD  
  $         #Ende des Subject  
)iDx
```

Lesbar? Verständlich? Wartbar?

Lesbare RegExp

```
(  
  ^           #Start des Subject  
  [a-z0-9_%. - ]+ #Benutzer  
  @           #Trennzeichen @  
  [a-z0-9. - ]+ #Domain  
  \.         #Trennzeichen .  
  [a-z]{2,4} #TLD  
  $         #Ende des Subject  
)iDx
```

- Zeilenumbrüche an beliebigen Stellen

Lesbare RegExp

```
(  
  ^           #Start des Subject  
  [a-z0-9_%. - ]+ #Benutzer  
  @           #Trennzeichen @  
  [a-z0-9. - ]+ #Domain  
  \.         #Trennzeichen .  
  [a-z]{2,4} #TLD  
  $         #Ende des Subject  
)iDx
```

- Alles nach einer Raute (#) bis zum Zeilenende wird ignoriert

Lesbare RegExp

```
(  
  ^           #Start des Subject  
  [a-z0-9_%. - ]+ #Benutzer  
  @           #Trennzeichen @  
  [a-z0-9. - ]+ #Domain  
  \.         #Trennzeichen .  
  [a-z]{2,4} #TLD  
  $         #Ende des Subject  
)iDx
```

- Alle Leerzeichen werden ignoriert es sei denn sie sind escaped (\)

POSIX

VS.

PCRE

POSIX vs. PCRE

- POSIX Basic Regular expressions bilden den Standard, der von den meisten *nix Systemtools akzeptiert wird
- PCRE oder ein Subset davon wird von nahezu allen modernen Hochsprachen unterstützt

POSIX: Delimiter

/regexp/

- Delimiter dürfen keine Klammern sein
- Identisches Zeichen auf beiden Seiten
- Viele Tools benötigen keinen Delimiter

POSIX: Subpattern

`\(subpattern\)`

- Subpattern müssen mit vorangestelltem Backslash (\) aktiviert werden
- Ohne diese Aktivierung werden die Klammern als Literal verarbeitet

POSIX: Subpattern Referenzen

`\(foo\)1`

- Subpattern-Referenzen sind auf 9 begrenzt
- Viele Tools erlauben jedoch mehr

POSIX: Number Quantifier

`A\{2,3\}`

- Number Quantifier müssen mit einem Backslash (\) aktiviert werden
- Ohne Aktivierung werden diese als Literal verarbeitet

POSIX: One or more - Quantifier

`a\+`

- Der + Quantifier ist kein Bestandteil von POSIX Regexp
- Viele Tools unterstützen ihn jedoch, wenn er mit vorangestelltem Backslash (\) aktiviert wird
- Ohne Aktivierung wird das + als Literal gewertet

POSIX: One or none - Quantifier

a\?

- Der ? Quantifier ist kein Bestandteil von POSIX Regexp
- Viele Tools unterstützen ihn jedoch, wenn er mit vorangestelltem Backslash (\) aktiviert wird
- Ohne Aktivierung wird das ? als Literal gewertet

POSIX: Choice

`a\|b\|c`

- Der Choice Operator (|) ist kein Bestandteil von POSIX Regexp
- Viele Tools unterstützen ihn jedoch, wenn er mit vorangestelltem Backslash (\) aktiviert wird
- Ohne Aktivierung wird das | Zeichen als Literal gewertet

POSIX: Zeichenklassen

`[:alnum:] [:alpha:] [:space:]`

- Es existieren spezielle vordefinierte Zeichenklassen in POSIX Regexp
- PCRE Klassen wie `\w`, `\s` und `\d` werden nicht unterstützt

POSIX: Erweiterte Fähigkeiten

- Keine Named Subpattern
- Keine Veränderung der Greediness
- Kein Unicode Support
- Keine Unterstützung für Subpattern Optionen
- Keine Unterstützung vieler Modifier
- Keine Assertions
- ...

Unicode

Unicode

UTF-8 Mode

- Modifier **u**

(**^abcdef\$**)**u**



Valides UTF-8 in Pattern und Subject erforderlich

Unicode

● UTF-8 Encoding

- Byte für alle ASCII Zeichen (0-127) identisch
- 2-4 Byte für weitere Zeichen (Codepoints)

● Codepoints

- Behandlung als ein Zeichen

Unicode

Русский

Unicode

`(\x{0420})u`

Русский

Unicode

`(\x{0420})u`

Русский



- `\x` Escape zur Angabe einzelner Codepoints

Unicode



`([\x{0400} - \x{A697}]+) u`

Русский

- `\x` Escape zur Angabe einzelner Codepoints
- Funktioniert in Zeichenklassen

Unicode



`(\p{Cyrillic}+)u`

Русский

- `\x` Escape zur Angabe einzelner Codepoints
- Funktioniert in Zeichenklassen
- Es gibt vordefinierte Zeichenklassen

Unicode

`(\p{Cyrillic}+)u`

Русский 한국어



- `\x` Escape zur Angabe einzelner Codepoints
- Funktioniert in Zeichenklassen
- Es gibt vordefinierte Zeichenklassen

Unicode



`(\p{L}+)`

Русский 한국어

- `\x` Escape zur Angabe einzelner Codepoints
- Funktioniert in Zeichenklassen
- Es gibt vordefinierte Zeichenklassen

Danke für Eure Aufmerksamkeit!

Fragen?

<http://joind.in/3882>

Jakob Westhoff

Mail: jakob@php.net

Twitter: [@jakobwesthoff](https://twitter.com/jakobwesthoff)