Regular Expressions – Doing magic with text

International PHP Conference

Frankfurt/Mörfelden, 7th of November 2007

About me

Kore Nordmann

- Studying computer science at the University Dortmund
- Working for eZ systems on eZ components
- Maintainer and / or Developer in multiple open source projects: Image_3D, KaForkl, ezcGraph, Torii, Busimess, PHPUnit, WCV, ...

Goal:

- Parse BBCodes
  - Example: "`Hello [b]world[/b]!`"
  - Simplified XML-like markup language commonly used in forums.
    - Uses [] instead of <>
    - Different attribute notation

(I do not want to discuss the sense of using it, here)

Agenda:

A **short** overview

The required basics

Matching recursive structures

Regular expressions

- In computer science:
  - Commonly tell you if a word (string) is element of a language (set of strings)
  - Matching only regular languages
- In practice:
  - Check if some input string matches some pattern
  - Tokenizing

Regular expressions in PHP (1)

- POSIX compatible regular expressions
- Represented by the ereg*() functions
  - Also used by some other functions like split()
  - Available in some extemsions like mb_ereg_*()
- Slow, featureless, deprecated, will be moved to PECL

Regular expressions in PHP (2)

- Perl compatible regular expressions (PCRE)
- Represented by the pcre_*() functions
- Fast, mighty, ... better
  - Also supports less mighty POSIX regular expressions

Agenda:

A **short** overview

The required basics

Matching recursive structures

Simple first expression

We use PCRE, and do not bother with POSIX compatible regular expression

Match an opening BBCode bold tag: `[b]`

```
(\[b\])i
```

Delimiters

Regexp: `(\[b\])i`

You may use a lot of different delimiters (except the backslash and alphanumeric characters)

- Common delimiters: /, #, @
- My personally preferred delimiters: ()
  - No escaping in the regular expression
  - Intuitive match count. (More later)

Pattern modifiers

- Regexp: `(\[b\])i`
- Alphanumeric characters appended after the closing delimiter
- Modify the behaviour of the regular expression
  - Common modifiers:
    - i: Case insensitive
    - S: Perform optimizations for non-anchored expressions
    - u: Pattern string is UTF-8
    - U: Ungreedy (explained later)
    - s: . matches all (explained later)
    - m: multiline (explained later)

Using regular expression with PHP

Regexp: `(\[b\])i`

PHP-Code:

```
<?php
  var_dump( preg_match(
      '(\\[b\\])i', 'Hello [b]world[/b]!'
  ) ); ?>
```

=> int(1)

Why use so many backslashes?

Getting back the matches

Regexp: `(\[b\])i`

PHP-Code:

```
<?php preg_match(
        '(\\[b\\])i', 'Hello [b]world[/b]!',
        $matches );
  var_dump( $matches ); ?>
```

```
=> array(1) { [0]=> "[b]" }
```

Matching all BBCodes

- Regexp: `(\[[a-z]+\])i`
- Character classes to defined sets of characters
  - Invert matching by `^` at the beginning: `[^a-z]`
  - Define ranges by using `-`
  - Most special chars are no special chars in character classes: `[()]`
- A number span would not look like: `[13-68]`, but: `(1[3-9]|[2-5][0-9]|6[0-8])`

Backreferences

Regexp: `(\[([a-z]+)\](.*)\[/\1\])is`

Matches: array(0 => ..., 1 => 'b', 2 => 'world')

Modifiers S & M

Multiline string:

```
Hello
  world!
```

```
(^(.+)$)    => NULL
(^(.+)$)m   => array( 'Hello', 'world!' )
(^(.+)$)ms  => array( 'Hello world!' )
(^(.+)$)s   => array( 'Hello world!' )
```

More modifiers: U

Input: `[b]Hello[/b] [b]world[/b]!`

Regexp: `(\[([a-z]+)\](.*)\[/\1\])is`

Matches:
array(0 => ..., 1 => 'b', 2 => 'Hello[/b] [b]world')

Regexp: `(\[([a-z]+)\](.*)\[/\1\])is`**U**

Matches:
array(0 => ..., 1 => 'b', 2 => 'Hello')

You may also use .*? to make some sub-expression ungreedy / greedy

You may use *preg_match_all()* to get all matches

Optional parameters

Input:

```
[url=http://kore-nordmann.de/blog/why_are_you_using_bbcod
es.html]Look here![/url]
```

Regexp:

```
(\[([a-z]+)(?:=([^\]]+))?\](.*)\[/\
1\])is
```

Matches:

```
array(
    0 => ..., 1 => "url", 2 =>
"http://kore[...].html", 3 => "Look here!"
)
```

Named matches

Input:

```
[url=http://kore-nordmann.de/blog/why_are_you_using_bbcod
es.html]Look here![/url]
```

Regexp:

```
(\[(?P<code>[a-z]+)(?:=(?
P<parameter>[^\]]+))?\](?P<content>.*)\[/\
1\])is
```

Matches:

```
array(
    0 => ..., 'code' => "url", 'parameter' =>
"http://kore[...].html", 'content' => "Look
here!"
)
```

Subpattern assertions

Input: `[b]Hello[/b] [cmsobject=12]world[/cmsobject]!`

Regexp: `(\[(?<!cms)([object]+)(?:=([^\]]+))?\](.*)\[/\1\])is`

Matches: ?

Other assertions:

`foo(?!bar)`	foo NOT followed by bar.
`for(?=bar)`	foo followed by bar.
`(?<!foo)bar`	bar NOT preprended by foo
`(?<=foo)bar`	bar preprended by foo

Conditional Subpatterns

Regexp: `(\[([a-z]+)(?:=((?(?=ftp)ftp://[^@]+@[^\]]+|[^\]]+))?\](.*)\[/\1\])i`

Requires a username (and password) for all parameters with FTP URLs

Example code

Agenda:

A **short** overview

The required basics

Matching recursive structures

Limitations of regular expressions

- Regular expressions neither know back references, nor subpattern modifiers
- Regular expressions can NOT match recursive structures.
  - Really.
- The very simple language: "**n opening braces, followed by n closing braces**" can not be validated

The mathematical proof

See:

http://kore-nordmann.de/blog/do_NOT_parse_using_regexp.html#id13

PCRE are no regular expressions

- PCRE implements a superset of regular expressions
- The language type matched by PCRE is not yet known for sure
  - Try to implement a Turing Machine using PCRE
- Matching the above language is possible with PCRE:

```
(\((((?>[^()]+)|(?R))*)\))
```

Matching recursive BBCode structures

Input: `Some [b] longer [i]text[/i][/b].`

Regexp:

```
(([^\[\]]*\[([a-z]+)(?:=(?
  [^\]]+))?\](?>[^\[\]]*|(?R))\[/\2\]
  [^\[\]]*))i
```

Validates correct BBCode structures

Example code...

Summary

- You should NOT use regular expressions to try such things
  - It is nearly impossible to debug
  - It is not maintainable at all
- You should use (PCRE) regular expressions for:
  - Matching patterns not structures
  - Tokenize

Questions?

Open questions?

Ressources:

http://php.net/pcre

http://kore-nordmann.de

The end

Thanks for listening