Komplexe reguläre Ausdrücke

Kürzlich habe ich mich mit HTML-Parsing beschäftigt und bin auf einen regulären Ausdruck gestoßen:

`/([\w-:\*>]*)(?:\#([\w-]+)|\.([\w-]+))?(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?([\/, ]+)/is`

(Dies ist ein regulärer Ausdruck in PHP, der nicht übersetzt werden muss, da er spezifische Muster in Zeichenketten sucht und keine sprachspezifischen Begriffe enthält.)

Es wird verwendet, um CSS-Selektoren abzugleichen, wie z.B. `div > ul`.

Früher bin ich oft auf solch komplexe Ausdrücke gestoßen und habe instinktiv zurückgeschreckt. Heute werde ich sie endlich verstehen! Ein Mann sollte sich selbst härter anpacken!

Übereinstimmung mit `div > ul`

Ich habe eine Website gefunden, https://regex101.com/, die es ermöglicht, online Regex-Muster zu testen und bietet auch Erklärungen dazu.

Obwohl die Erklärung auf der rechten Seite einige Klarheit gebracht hat, ist immer noch nicht ganz klar, wie die genaue Übereinstimmung funktioniert. Also nehmen wir ein paar Beispiele und analysieren sie einzeln.

Der spezifische Code, in dem dieser reguläre Ausdruck auftaucht, ist:

```
$matches = [];
preg_match_all($this->pattern, trim($selector).' ', $matches, PREG_SET_ORDER);
```

Hinweis: Der Code wurde nicht übersetzt, da es sich um eine Programmiersprache handelt, die in der Regel nicht übersetzt wird.

`preg_match_all` bedeutet, alle Zeichenketten zu erhalten, die dem Muster entsprechen. Wenn es gibt:

```
preg_match_all("abc", "abcdabc", $matches)
```

(Der Code bleibt unverändert, da es sich um eine PHP-Funktion handelt, die in der Regel nicht übersetzt wird.)

Der erste Parameter ist das Muster, der zweite Parameter ist die Zeichenkette, die abgeglichen werden soll, und der dritte Parameter ist die Ergebnisreferenz. Nach der Ausführung enthält das `$matches`-Array zwei `abc`.

Mit diesem Verständnis im Hinterkopf passt der Ausdruck `div > ul` im obigen Bild nur auf die ersten vier Zeichen `div >`. Unterstützt `regex101` nicht `preg_match_all`? Zum Glück reicht es aus, einen Modifikator namens `g` hinzuzufügen:

Durch das Hinzufügen von `g` wird alles übereinstimmende gefunden, anstatt beim ersten Treffer zurückzukehren.

Nachdem wir es hinzugefügt haben, haben wir `div > ul` abgeglichen:

Auf der rechten Seite ist zu sehen, dass im ersten Match, also `div`, wir mit den Regeln der ersten Gruppe `div` gematcht haben und dann mit den Regeln der siebten Gruppe das Leerzeichen gematcht haben.

Lassen Sie uns nun die Erklärung der ersten Gruppe von Regeln betrachten:

In diesem langen Ausdruck wird der erste Klammerausdruck als erste Gruppe von Regeln bezeichnet. Dies ist eine Erfassungsgruppe. Die Klammern selbst werden nicht abgeglichen, sondern dienen der Gruppierung. `[]` steht für eine Zeichenmenge, und die darin enthaltenen Regeln beschreiben, wie diese Zeichenmenge aussieht. Diese Zeichenmenge enthält:

- `\w` steht für Groß- und Kleinbuchstaben, die Zahlen von 0 bis 9 sowie den Unterstrich.
- `-:` repräsentiert direkt diese beiden Zeichen in der Menge.
- `\*` da `*` ein reserviertes Zeichen in regulären Ausdrücken ist und eine spezielle Bedeutung hat, muss es mit `\` maskiert werden, um ein normales `*`-Zeichen darzustellen.
- `>` stellt einfach das Zeichen `>` dar.

`[\w-:\*>]*` □□□ `*` □□□□□□□□□□□ 0 □□□□□□□□□□□□□□□□□□□□□□□□ `div` □□□ `\w` □□□ `d`□`i`□`v`□□□□□ □□□□□□□□□□□□□□□□□□□□□□ `[]` □□ `(?:)`□□□□ `([\w-:\*>]*)` □□□□□□□□□□□□□□□ `(?:[\w-:\*>]*)`□

Warum dann nicht einfach keine Klammern verwenden, wenn sie nicht im Ergebnis erscheinen sollen? Klammern dienen der Gruppierung, und Gruppierung ist durchaus sinnvoll. Weitere Informationen finden Sie unter „What is a non capturing group? (?:) - StackOverflow“.

Nachdem wir besprochen haben, dass `div` die erste Gruppe von Regeln erfüllt, wollen wir nun erklären, warum das Leerzeichen die Regeln der siebten Gruppe erfüllt.

`[\/, ]` bedeutet, dass eines dieser vier Zeichen übereinstimmt, und `+` bedeutet, dass das vorherige Muster ein- oder mehrmals auftritt, wobei die Anzahl der Übereinstimmungen so groß

wie möglich sein soll. Da diese vier Zeichen ein Leerzeichen enthalten, wird unser Leerzeichen übereinstimmen. Da das nächste Zeichen nach `div` ein `>` ist, wird die Regel der siebten Gruppe nicht mehr erfüllt und die Übereinstimmung wird nicht fortgesetzt.

Ich habe verstanden, wie die Übereinstimmung von `div` funktioniert. Aber warum haben die Regeln der zweiten bis sechsten Gruppe die Leerzeichen hier nicht erfasst, sondern sie der siebten Gruppe überlassen?

Erklärung zum zweiten Teil:

Zunächst bedeutet `(?:)`, dass dies eine nicht erfassende Gruppe ist. Das `?` am Ende zeigt an, dass das vorherige Muster 0 oder 1 Mal auftreten kann. Daher kann der obige Ausdruck `(?:\#([\w-]+)|\.([\w-]+))?` vorhanden sein oder nicht. Wenn wir die äußeren Modifikatoren entfernen, bleibt `\#([\w-]+)|\.([\w-]+)` übrig, wobei das `|` in der Mitte ein "oder" darstellt, was bedeutet, dass eines der beiden Muster erfüllt sein muss. In `\#([\w-]+)` passt `\#` auf das `#`-Zeichen, und `[\w-]+` passt auf andere Zeichen. Im zweiten Teil passt `\.([\w-]+)` auf das `.`-Zeichen.

Daher können die Gruppen 2 bis 6 möglicherweise nicht erfüllt werden, da Leerzeichen nicht die erforderlichen Anfangszeichen für diese Gruppen sind. Da diese Gruppen jedoch einen `?`-Modifikator haben, ist es in Ordnung, wenn sie nicht erfüllt werden, und daher wird zur siebten Gruppe gesprungen.

Der `>` nach `div > ul` bleibt gleich:

Die erste Regelgruppe `([\w-:\*>]*)` hat `>` erfasst, und die siebte Regelgruppe `([\/, ]+)` hat ein Leerzeichen erfasst. Anschließend wird `ul` wie `div` behandelt.

Übereinstimmung mit `#answer-4185009 > table > tbody > td.answercell > div > pre`

Als nächstes kommt ein etwas komplexerer Selektor `#answer-4185009 > table > tbody > td.answercell > div > pre` (du kannst auch https://regex101.com/ öffnen und dies dort einfügen, um es zu testen):

Dies ist ein Kopieren und Einfügen aus Chrome:

Erste Übereinstimmung:

Da in der ersten Gruppe die Zeichenmenge `([\w-:\*>]*)` innerhalb der `[]` kein Zeichen enthält, das mit `#` übereinstimmt, und das abschließende `*` die Möglichkeit bietet, 0 oder mehr Vorkommen zu finden, wird hier 0 Mal übereinstimmen. Anschließend beschreibt die zweite Regel:

Wie bereits oben analysiert wurde, schauen wir uns direkt den Teil vor dem | an, nämlich \#([\w-]+). Hier wird \# verwendet, um das # zu matchen, und [\w-]+ matcht answer-4185009. Der darauf folgende Teil \.([\w-]+) würde angewendet werden, wenn es sich um .answer-4185009 handeln würde.

Als nächstes betrachten wir die Übereinstimmung td.answercell,

Die erste Gruppe der Regel ([\w-:*>]*) hat td abgeglichen, während der hintere Teil des zweiten großen Abschnitts (?:\#([\w-]+)|\.([\w-]+))?, nämlich \.([\w-]+), .answercell abgeglichen hat.

Die Analyse dieses Selektors endet hier.

Übereinstimmung mit a[href="http://google.com/"]

Als Nächstes passen wir den Selektor a[href="http://google.com/"] an:

Schauen wir uns den dritten großen Block an:

Der dritte große Ausdruck lautet (?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?. Zunächst bedeutet das äußerste (?:), dass es sich um eine nicht erfassende Gruppe handelt. Das ? am Ende zeigt an, dass dieser gesamte Block 0 oder 1 Mal vorkommen kann. Wenn wir dies entfernen, erhalten wir \[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\]. \[entspricht dem Zeichen [. @? bedeutet, dass das Zeichen @ optional ist. Die nächste Gruppe (!?[\w-:]+) zeigt an, dass das Zeichen ! optional ist, und [\w-:]+ entspricht href. Die folgende Gruppe (?:([!*^$]?=)["']?(.*?)["']?) ist eine nicht erfassende Gruppe. Wenn wir die äußerste Schicht entfernen, erhalten wir ([!*^$]?=)["']?(.*?)["']?. Hier bedeutet ([!*^$]?=), dass [!*^$]? 0 oder 1 Zeichen aus den eckigen Klammern [] entspricht. Dann folgt das direkte Zeichen =. Anschließend entspricht ["']?(.*?)["']? dem Ausdruck "http://google.com/". ["']? bedeutet, dass entweder " oder ' oder keines von beiden übereinstimmt. Wenn wir diese äußerste Schicht entfernen, erhalten wir (.*?), was http://google.com/ entspricht. Hier bedeutet *?, dass so wenig wie möglich übereinstimmen soll, d.h., wenn ein " oder ' vorhanden ist, soll es dem nachfolgenden Ausdruck ["']? überlassen werden. Daher wird nicht http://google.com/" übereinstimmen, sondern nur http://google.com/. Somit entspricht der gesamte Selektor a[href="http://google.com/.

”]‘ beendet die Übereinstimmung.

Zusammenfassung

Endlich verstanden! Lassen Sie uns das noch einmal klarstellen. Zunächst besteht der gesamte komplexe Ausdruck `([\w-:\*>]*)(?:\#([\w-]+)|\.([\w-]+))?(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?([\/, ]+)` aus vier Hauptteilen:

- `([\w-:\*>]*)`
- `(?:\#([\w-]+)|\.([\w-]+))?`
- `(?:\[@?(!?[\w-:]+)(?:([!*^$]?=)["']?(.*?)["']?)?\])?`
- `([\/, ]+)`

(Dieser Codeblock enthält reguläre Ausdrücke, die nicht übersetzt werden sollten, da sie spezifische Muster für die Textverarbeitung darstellen.)

Der komplexeste dritte Teil besteht wiederum aus diesen Komponenten:

- `\[`
- `(!?[\w-:]+)`
- `(?:([!*^$]?=)["']?(.*?)["']?)?`
- `\]`

Daher können all diese ausreichend kleinen Teile einzeln angegangen werden. Dann sucht man sich ein paar Beispiele mehr, schaut sich an, wie jedes Beispiel passt, und analysiert es gleichzeitig mit der Erklärung auf https://regex101.com/. So hat man diesen scheinbar komplexen regulären Ausdruck verstanden – es stellt sich heraus, dass er ein Papiertiger ist!