# ReDoS に負けない正規表現づくり

鳥海塾プログラミング班　澤田 羽衣　鶴岡工業高等専門学校　2 年

## 1. 目　的

ReDos（Regular expression Denial of Service）は VM 型の正規表現エンジンで起こる、正規表現の弱点を突く DoS 攻撃である。これにより実際に Web サービスが DoS 状態となった実例もある [1][2]。主な原因は過剰なバックトラックであるが、これは正規表現の書き方が原因で起こる。各種プログラム言語でも対策が講じられつつあるが、言語のアップデートだけではカバーできていない場合や、対策前の言語があるのも事実である。そこで本研究では、正規表現の書き方を工夫することで明日からできる対策を提示する。

## 2.「だめな」正規表現

ここでは、書き方が原因で弱点を持つ正規表現を「だめな」正規表現と呼ぶ。「だめな」正規表現が弱点を突かれると、途端にマッチ試行に人間にも長いと感じられるほど多くの時間を費やす。このとき、外から見ればプログラムはまるごとフリーズしている。

## 3. 遅くなりやすい書き方

| `/(a+)+$/` | `/^(a\|a)*$/` | `.*` |
|---|---|---|
| のように多重の量指定子があるもの | のように冗長な部分があるもの | が多く使われており曖昧なもの |

これらの書き方を用いた正規表現では、マッチしそうな長い文字列が弱点となりやすい [3]。

## 4. 解決策

**バックトラックを減らすよう意識する**

バックトラック…文字列の途中まで一旦マッチしたが、その先のマッチで失敗した場合、マッチした文字の一部を手放しそこから再試行する動作

### ▪ 強欲な量指定子を使う

普段使われる貪欲な量指定子に「+」を付加した形をしている（ 例えば /.*/ の強欲な形は /.*+/）、バックトラックをしない最大量指定子。これを使うと検索が難しくなるが、使ったうえで任意の文字列にマッチさせられるように書けば高速なパターンが得られる。強欲な量指定子が無い言語もあるが、一度強欲な量指定子を用いて書いたパターンから強欲のプラスを除いたものを用いると、強欲を用いた場合に近いパフォーマンスが得られることが多い。

### ▪ 怠惰な量指定子を使う

普段使われる貪欲な量指定子に「?」を付加した形をしている。最短結果にマッチする。貪欲な量指定子ではマッチしすぎる場合に使われる。

### ▪ アトミックグループを使う

アトミックグループとして囲った範囲のパターンはマッチに失敗してもバックトラックをしない。「(?>pattern)」といった書き方をする。

実際に ReDoS が起こりそうな例として、Markdown 記法の検出に使う正規表現を取り上げる。Markdown は HTML に変換される記法で、その中に「ハイフンとスペースだけで構成され、かつハイフンが 3 つ以上含まれる行は HTML の <hr> に変換される」というルールがある。これを満たす行にマッチする正規表現として /^( *-+ *){3,}$/ というものが考えられる。しかし、これは量指定子が重なっており、マッチ時間が指数的に伸びてしまう [3]。これの量指定子を強欲なものに置き換えるとバックトラックがなくなり、マッチに要する step 数が大幅に減る [4]。step 数の確認には Regex101 というサイトを使用した。

`/^( *-+ *){3,}$/`

| 文字列 | step 数 |
|---|---|
| 「 -　- --」 | 18 |
| 「----------　-」 | 52,108 |

https://regex101.com/r/zWnhFy/1

`/^( *+-++ *+){3,}+$/`

| 文字列 | step 数 |
|---|---|
| 「 -　- --」 | 22 |
| 「----------　-」 | 53 |

https://regex101.com/r/zWnhFy/2

### 実　測　「だめな」正規表現と工夫した正規表現で本当にマッチ時間に差が出るのか？

/usr/share/dict/words の中の 4 ～ 7 文字の単語

ダミー URL 生成　　ダミー URL 生成

拡張子付き 10,000 行　　拡張子なし 10,000 行

5,000 行ずつサンプル

シャッフルした 10,000 行を 10 回複製

パス部分の階層の深さを変えて計測

▲ 計測作業はシェルスクリプトで自動化した

計測を行った言語

| 言語 | バージョン |
|---|---|
| Ruby | 3.1.4 |
| JavaScript (node.js) | v10.19.0 |

・拡張子で終わる URL にマッチする 2つの正規表現

「だめな」正規表現
`/(https|http):\/\/.*\/.*\..*/`

工夫した正規表現
`/https?:\/\/[^\/]+\/[^.\n]*\.[^\n ,.]+/`

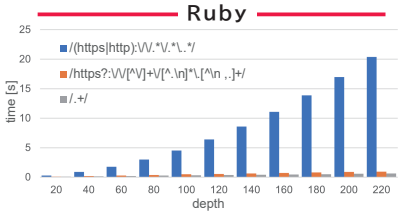
※考案時は /https?:\/\/[^\/]++\/[^.\n]*+\.[^\n ,.]++/としていたが、JavaScriptに強欲な量指定子がないためパフォーマンスへの影響を確認したうえで貪欲な量指定子に変更した。

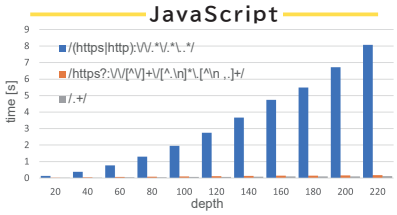
・単純に全行の走査をする正規表現

`/.+/`

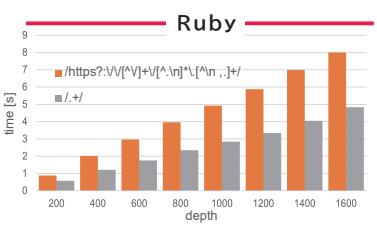
http://www.abalone-runoff.biz/Sucre/miserly/reports/whose/Blythe/wording/flyover/adeptly/Armenia/fronted.png

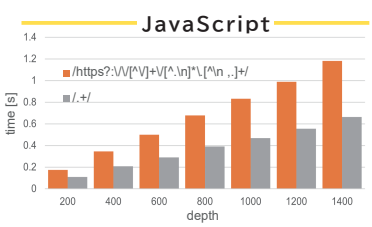
▲ 10 階層のダミー URL




どちらの計測でも階層を深くしていくと「だめな」正規表現は大きく低速になったが、工夫したパターンではマッチにかかる時間が大きく短縮された。工夫した正規表現が本当に遅くなりにくくなっているのかも検証するため、「だめな」正規表現を除いた2つのパターンについて追加で計測を行った。




※Rubyは1600階層まで計測したが、JavaScriptではNode.jsの文字列長の上限を超えてしまい1600階層の計測は断念した。
※実測に使用したパターンの書き方には文献 [5]，[6] を参考にした。

工夫した正規表現と走査の正規表現は、階層を増やすと線形にマッチ時間が伸びていった。

## 5. まとめ・考察・展望

強欲な量指定子が使える場合は、それを使ってバックトラックに頼らない書き方をした方が安全な形になる。今回計測した、末尾に拡張子のある URL にマッチする正規表現では「だめな」書き方をすると $O(n^x)$ $(1 < x < 2)$ となったが、工夫して書くと線形時間でマッチさせることができた。バックトラックを抑える形に工夫することで速度低下しづらいパターンが作れたと考えられる。

今回は強欲な量指定子を活用したアプローチにより、マッチングを高速化することができた。ただ強欲な量指定子が使えない言語も存在するため、強欲な量指定子を貪欲な量指定子に書き直してもパフォーマンスに問題が生じないかを調べられる静的解析ツールを作りたい。

参考文献

1. 藤浪大弥、監修：牧大輔ほか．第 74 回　正規表現の脆弱性「ReDoS」徹底解説 ～原理と対策から、Perl での最適化まで（1）．https://gihyo.jp/dev/serial/01/perl-hackers-hub/007401,（参照 2023-10）.
2. JVN. JVN#86484824: シングラフィカ製メールフォームプロ CGI における正規表現を用いたサービス運用妨害（ReDoS）の脆弱性 .https://jvn.jp/jp/JVN86484824/,（参照 2023-11-09）.
3. James Davis. The Regular Expression Denial of Service (ReDoS) cheat-sheet. https://levelup.gitconnected.com/the-regular-expression-denial-of-service-redos-cheat-sheet-a78d0ed7d865,（参照 2023-11）.
4. 新原　雅司．パフォーマンスを意識して正規表現を書く .(2022 年参照 ) https://blog.shin1x1.com/entry/regex-performance
5. RexEgg.The Best Regex Trick. https://www.rexegg.com/regex-best-trick.html,（参照 2023-09）.
6. Liz Bennett. Regexes: The Bad, the Better, and the Best. https://www.loggly.com/blog/regexes-the-bad-better-best/,（参照 2022）.

本研究は東北公益文科大学ジュニアリサーチャー制度の支援を受けています。