**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

Направление (специальность) ⎯ 09.04.04 Программная инженерия

Специализация ⎯ Веб-технологии

Дисциплина — Инновационные исследования в вебе

Курсовой проект (работа)

ТЕМА: Анализ зависимости времени выполнения регулярных выражений от их сложности на платформе Node.JS

ВЫПОЛНИЛ

Студент группы P41071 Симаненко Ю.

№  группы ФИО

ПРОВЕРИЛ доцент, кандидат педагогических наук Государев И. Б.

ученая степень, должность ФИО

САНКТ-ПЕТЕРБУРГ  
2022 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc106218502)

[ХОД ИССЛЕДОВАНИЯ 3](#_Toc106218503)

[РЕЗУЛЬТАТЫ 6](#_Toc106218504)

[ВЫВОДЫ 6](#_Toc106218505)

[Заключение 7](#_Toc106218506)

[Источники 7](#_Toc106218507)

[ОТЗЫВ РУКОВОДИТЕЛЯ 1](#_Toc106218508)

ВВЕДЕНИЕ

Регулярные выражения — это средство обработки и разбора текста по заданному набору правил. Они являются популярным инструментом для решения множества современных практических задач, связанных с валидацией текста или извлечением необходимых данных из строк по заранее определенному шаблону.[1]

Одним из недостатков регулярных выражений с точки зрения разработчика является сложность оценки времени их работы. Данная проблема возникает из-за того, что время исполнения регулярного выражения зависит не только от шаблона, но и от входной строки, а также от внутренних особенностей конкретной реализующей их библиотеки [2]. Тем не менее, если знать, как языковые конструкции регулярных выражений влияют на скорость выполнения операции поиска по шаблону, можно разработать рекомендации и для конкретного случая выбирать лучшее решение, корректируя исходный шаблон.

Целью данной статьи является выявление конструкций регулярных выражений, замедляющих скорость выполнения операций поиска по шаблону в строке на платформе Node.js.

ХОД ИССЛЕДОВАНИЯ

Для того, чтобы результаты экспериментов можно было корректно сравнивать между собой, необходимо, чтобы и входной текст, и результат сопоставления были одинаковы для всех сравниваемых выражений. В связи с этим, было выбрано одно исходное выражение, реализующее следующий набор правил:

* первое слово начинается с заглавной буквы;
* остальные буквы в первом слове — только строчные;
* максимальная длина первого слова равна пяти символам;
* все слова в предложении разделены только пробелами;
* все слова кроме первого состоят только из строчных букв;
* искомая подстрока заканчивается точкой.

Кроме того, предполагается, что все буквы во входных строках являются латиницей. Поддержка букв из других алфавитов не требуется, но и не запрещается.

Исходное выражение, удовлетворяющее ранее перечисленным требованиям, выглядит следующим образом: “[A-Z][a-z]{1,4}(\s[a-z]+)+\.”

Все остальные используемые для экспериментов регулярные выражения являются модификациями исходного выражения [3], при этом удовлетворяют предъявленным к исходной строке требованиям.

Выбран следующий набор регулярных выражений:

|  |  |  |
| --- | --- | --- |
| Название теста | Регулярное выражение | Пояснения |
| Origin | [A-Z][a-z]{0,4}(\s[a-z]+)+\. | Исходное регулярное выражение |
| LetterDiap | [ABCDEFGHIJKLMNOPQRSTUVWQYZ][abcdefghijklmnopqrstuvwxyz]{0,4}(\s[abcdefghijklmnopqrstuvwxyz]+)+\. | Диапазоны букв заменены перечислением |
| Unicode | \p{Lu}\p{Ll}{0,4}(\s\p{Ll}+)+\. | Помимо латинских букв, выражение ищет и любые другие буквы, поддерживаемые в юникоде |
| DiapToQuestions | [A-Z][a-z]?[a-z]?[a-z]?[a-z]?(\s[a-z]+)+\. | Численный диапазон {0, 4} заменен на написанную четыре раза конструкцию **[a-z]?** |
| SToSpaces | [A-Z][a-z]{0,4}( [a-z]+)+\. | Вместо выражения \s для поиска пробелов используется символ пробела |

Таблица 1— Регулярные выражения, используемые в тестах

Для проверки каждого регулярного выражения использовались три строки разного размера, каждая была помещена в файл.

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Описание | Размер | Длина строки в символах |
| small.txt | Строка состоит из одного короткого предложения, удовлетворяющего поставленным условиям | 44 b | 44 |
| middle.txt | Строка сгенерирована и состоит из простых предложений, удовлетворяющих и не удовлетворяющих условиям | 20 Kb | 19895 |
| large.txt | Строка является частью реально существующего литературного произведения и состоит из простых и сложных предложений | 186 Kb | 190564 |

Таблица 2 — Описание входных строк

Все тесты выполнялись в следующем окружении:

* Версия Node.js: v14.15.3
* ОС: 64-разрядная Windows 10 (сборка 19042.1237, версия 20H2)
* Процессор: Intel Core i7-8550U @ 1.80GHz
* Оперативная память: 16 GB DDR4

Все регулярные выражения выполнялись с помощью функции String.prototype.match() [4][5] с флагами «g» и «u». Исходный код тестов, а также файлы с входными строками доступны в Github-репозитории [6].

Каждый тест выполнялся 100 раз, среднее арифметическое всех полученных значений времени выполнения выбиралось в качестве усредненного значения времени выполнения соответствующего теста.

РЕЗУЛЬТАТЫ

В Таблице 3 представлено усредненное время выполнения 100 000 итераций [7] с входной строкой из соответствующего файла (small.txt, middle.txt, large.txt).

|  |  |  |  |
| --- | --- | --- | --- |
|  | small.txt (мс) | middle.txt (мс) | large.txt (мс) |
| Origin | 31,2 | 5091 | 54183 |
| LetterDiap | 31,2 | 5399 | 49874 |
| Unicode | **33,3** | **23848** | 52939 |
| DiapToQuestions | 31,1 | 7196 | **87426** |
| SToSpaces | 30,5 | 5014 | 53062 |

Таблица 3 — Результаты времени поиска по регулярному выражению для разных входных строк

ВЫВОДЫ

В результате сравнения экспериментального времени выполнения поиска по тексту на приведенном наборе регулярных выражений было выявлено, что наиболее значительное ухудшение данного показателя вносила поддержка символов Unicode. Это может быть связано с необходимостью проверки большего количества символьных диапазонов [8], а также с особенностями представления символов в кодировке UTF-8 [9], использующей динамическое количество байт для записи символа в зависимости от его положения в данной таблице [10]. Кроме того, негативное изменение времени поиска вызывает замена квантификатора диапазона на записанную четыре раза конструкцию с квантификатором «один и более». Замена идентификатора символьного класса \s на символ пробела несущественно повышает производительность.

Исходя из вышеперечисленного, не рекомендуется использовать поддерживающие Unicode конструкции в случаях, когда этого не требуется для решения поставленной задачи. Для повторения выражения заранее известное количество раз следует использовать квантификатор диапазона.

Заключение

В ходе выполнения работы были составлены пять регулярных выражений, возвращающих одинаковый результат при выполнении поиска в заданных строках. Были экспериментально получены усредненные значения времени выполнения каждого выражения на каждом наборе тестовых данных. На основании полученных значений был проведен анализ и выявлены языковые конструкции, негативно влияющие на время исполнения выражения.

Источники

1. Форта Бен Изучаем регулярные выражения - пер. с англ. Берштейн И. В. — М.: Вильямс, 2019. - 192 с. - ISBN 978-5-6041394-2-4
2. David Mertz, Jay Trolinger, Sven Geier The Puzzling Quirks of Regular Expressions. — : Lulu Press, 2021. — 100 с. — ISBN 9781312160743
3. Regular Expression, IEEE Std 1003.1-2017, Open Group, 2018 edition [Электронный ресурс] URL: <https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html> (дата обращения: 16.10.2021)
4. Properties of the String Prototype Object // ECMAScript 2022 Language Specification [Электронный ресурс] URL: <https://tc39.es/ecma262/multipage/text-processing.html#sec-properties-of-the-string-prototype-object> (дата обращения: 16.10.2021).
5. Properties of the RegExp Prototype Object // ECMAScript 2020 Language Specification [Электронный ресурс] URL: <https://262.ecma-international.org/11.0/#sec-string.prototype.match> (дата обращения: 16.10.2021)
6. Github repository regExpPerformanceTest [Электронный ресурс] URL: <https://github.com/yuliasimanenko/regExPerformanceTest> (дата обращения: 16.10.2021)
7. Node.js Performance documentation [Электронный ресурс] URL: <https://nodejs.org/api/perf_hooks.html#perf_hooks_perf_hooks_performance> (дата обращения: 16.10.2021)
8. Любанович Билл Простой Node.js. Современный стиль программирования. — СПб.: Питер, 2016. — 429 с.: — (Серия «Бестселлеры O’Reilly»).
9. Лутц М. Программирование на Node.js, том III, 2-е издание. — пер. с англ. — СПб.: Символ-Плюс, 2016. — 511 с.
10. Государев И. Б. Введение в веб-разработку на языке JavaScript: Учебное пособие. — СПб.: Издательство «Лань», 2019. — 144 с. ISBN 978-5-8114-3539-5

**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет ИТМО»**

**(Университет ИТМО)**

**ОТЗЫВ РУКОВОДИТЕЛЯ**

**о выполнении курсового проекта (работы)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | | Симаненко Ю. | | | | | | |
| (Фамилия, И., О.) | | | | | | | | |
| Факультет | ПИиКТ | | | | | | Группа | P41071 |
| Направление (специальность) | | | | | | 09.04.04 «Программная инженерия» | | |
| Руководитель | | | | Государев И.Б., ИТМО ПИиКТ, доцент, кандидат педагогических наук | | | | |
| (Фамилия, И., О., место работы, должность, ученое звание, степень) | | | | | | | | |
|  | | | | | | | | |
| Дисциплина | | | Инновационные исследования в вебе | | | | | |
|  | | | | | | | | |
| Наименование темы: | | | | | Анализ зависимости времени выполнения регулярных | | | |
| |  | | --- | | выражений от их сложности на платформе Node.JS | | | | | | | | | |
|  | | | | | | | | |

**ОЦЕНКА КУРСОВОГО ПРОЕКТА (РАБОТЫ)**

**Отмеченные достоинства:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отмеченные недостатки:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Заключение:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_