

Раздел

[Регулярные выражения](#)

Навигация по уроку

Жадный поиск

Ленивый режим


Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться

[Редактировать на GitHub](#) → [Регулярные выражения](#) 13-го июня 2020

## Жадные и ленивые квантификаторы

На первый взгляд квантификаторы – это просто, но на самом деле это не совсем так.

Нужно очень хорошо разбираться, как работает поиск, если планируем искать что-то сложнее, чем `/\d+/`.

Давайте в качестве примера рассмотрим следующую задачу:

У нас есть текст, в котором нужно заменить все кавычки `"..."` на «ёлочки» `«...»`, которые используются в типографике многих стран.

Например: `"Привет, мир"` должно превратиться в `«Привет, мир»`. Есть и другие кавычки, вроде `„Witam, świat!”` (польский язык) или `「你好, 世界」` (китайский язык), но для нашей задачи давайте выберем `«...»`.

Первое, что нам нужно – это найти строки с кавычками, а затем мы сможем их заменить.

Регулярное выражение вроде `/"."/g` (кавычка, какой-то текст, другая кавычка) может выглядеть хорошим решением, но это не так!

Давайте это проверим:

```
1 let regexp = /".+"/g;
2
3 let str = 'a "witch" and her "broom" is one';
4
5 alert( str.match(regexp) ); // "witch" and her "broom"
```

...Как мы видим, регулярное выражение работает не как задумано!

Вместо того, чтобы найти два совпадения `"witch"` и `"broom"`, было найдено одно: `"witch" and her "broom"`.

Причину можно описать, как «жадность – причина всех зол».

### Жадный поиск

Чтобы найти совпадение, движок регулярных выражений работает по следующему алгоритму:

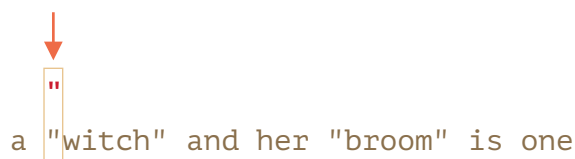
- Для каждой позиции в строке для поиска:
  - Попробовать найти совпадение с шаблоном на этой позиции.
  - Если нет совпадения, переход к следующей позиции.

Эти общие слова никак не объясняют, почему регулярное выражение работает неправильно, так что давайте разберём подробно, как работает шаблон `"."/`.

1. Первый символ шаблона – это кавычка `"`.

Движок регулярного выражения пытается найти его на нулевой позиции исходной строки `a "witch" and her "broom" is one`, но там – `a`, так что совпадения нет.

Он продолжает: двигается к следующей позиции исходной строки и пытается найти первый символ шаблона там. У него не получается, он двигается дальше и, наконец, находит кавычку на третьей позиции:



a "witch" and her "broom" is one

Раздел

Регулярные выражения

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



2. Кавычка замечена, после чего движок пытается найти совпадение для оставшегося шаблона. Смотрит, удовлетворяет ли остаток строки шаблону .+" .

В нашем случае следующий символ шаблона: . (точка). Она обозначает «любой символ, кроме новой строки», так что следующая буква строки 'w' подходит.

a "witch" and her "broom" is one

3. Затем точка повторяется из-за квантификатора .+. Движок регулярного выражения добавляет к совпадению один символ за другим.

...До каких пор? Точке соответствуют любые символы, так что движок остановится только тогда, когда достигнет конца строки:

a "witch" and her "broom" is one

4. Тогда он перестанет повторять .+ и попытается найти следующий символ шаблона. Это кавычка ". Но есть проблема: строка для поиска закончилась, больше нет символов!

Движок регулярного выражения понимает, что захватил слишком много .+ и начинает отступать.

Другими словами, он сокращает совпадение по квантификатору на один символ:

a "witch" and her "broom" is one

Теперь он предполагает, что .+ заканчивается за один символ до конца строки и пытается сопоставить остаток шаблона для этой позиции.

Если бы тут была кавычка, тогда бы поиск закончился, но последний символ – это 'e', так что он не подходит.

5. ...Поэтому движок уменьшает количество повторений .+ ещё на один символ:

a "witch" and her "broom" is one

Кавычка ' ' не соответствует 'n'.

6. Движок продолжает возвращаться: он уменьшает количество повторений '.', пока оставшийся шаблон (в нашем случае ' "') не совпадёт:

a "witch" and her "broom" is one

7. Совпадение найдено.

Раздел

Регулярные выражения

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



8. Так что первое совпадение: "witch" and her "broom". Если у регулярного выражения стоит флаг g, то поиск продолжится с того места, где закончился предыдущий. В оставшейся строке is one нет кавычек, так что совпадений больше не будет.

Это, определённо, не то, что мы ожидали. Но так оно работает.

**В жадном режиме (по умолчанию) квантификатор повторяется столько раз, сколько это возможно.**

Движок регулярного выражения пытается получить максимальное количество символов, соответствующих . +, а затем сокращает это количество символ за символом, если остаток шаблона не совпадает.

В нашей задаче мы хотим другого. И нам поможет ленивый режим квантификатора.

## Ленивый режим

«Ленивый» режим противоположен «жадному». Он означает: «повторять квантификатор наименьшее количество раз».

Мы можем включить его, вставив знак вопроса '?' после квантификатора, то есть будет \*? или +? или даже ?? для '?'.

Проясним: обычно знак вопроса ? сам по себе является квантификатором (ноль или один), но, если он добавлен *после другого квантификатора (или даже после самого себя)*, он получает другое значение – он меняет режим совпадения с жадного на ленивый.

Регулярное выражение /" .+?"/g работает как задумано, оно находит "witch" и "broom":

```
1 let regexp = /" .+?"/g;
2
3 let str = 'a "witch" and her "broom" is one';
4
5 alert( str.match(regexp) ); // witch, broom
```

Чтобы лучше понять, что поменялось, давайте рассмотрим процесс поиска шаг за шагом.

1. Первый шаг будет таким же: движок находит начало шаблона '"' на 3-ей позиции:

a "witch" and her "broom" is one

2. Следующий шаг аналогичен: он найдёт совпадение для точки '.':

a "witch" and her "broom" is one

3. А отсюда поиск продолжится по-другому. Из-за того, что у нас включён ленивый режим для +?, движок не будет пытаться найти совпадение для точки ещё раз, оно остановится и попробует найти совпадение для оставшегося шаблона '"' прямо сейчас:

a "witch" and her "broom" is one

Раздел

Регулярные выражения

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



Если бы на этом месте была кавычка, то поиск бы закончился, но там находится 'i', то есть совпадения нет.

4. Тогда движок регулярного выражения увеличит количество повторений для точки и попробует ещё раз:

a "witch" and her "broom" is one

Опять неудача. Тогда количество повторений будет увеличено ещё и ещё...

5. ...До тех пор, пока совпадение для оставшегося шаблона не будет найдено:

a "witch" and her "broom" is one

6. Следующий поиск начнётся с того места, где закончилось текущее совпадение и у нас будет ещё один результат:

a "witch" and her "broom" is one

В этом примере мы увидели, как ленивый режим работает для +? . Квантификаторы +? и ?? работают аналогичным образом – движок регулярного выражения увеличит количество совпадений, только если не сможет найти совпадение для оставшегося шаблона на текущей позиции.

**Ленивый режим включается только для квантификаторов с ? .**

Остальные квантификаторы остаются жадными.

Например:

```
1 alert( "123 456".match(/\d+ \d+?/) ); // 123 4
```

1. Шаблон \d+ пытается найти столько цифр, сколько возможно (жадный режим), так что он находит 123 и останавливается, потому что следующим символом будет пробел ' '.
2. Дальше в шаблоне пробел и в строке тоже, так что есть совпадение.
3. Затем идёт \d+?. Квантификатор находится в ленивом режиме, так что он находит одну цифру 4 и проверяет, есть ли совпадение для оставшегося шаблона с этого места.

...Но в шаблоне \d+? больше ничего нет.

Ленивый режим ничего не повторяет без необходимости. Шаблон закончился, заканчивается и поиск. Мы получаем 123 4.

Раздел

Регулярные выражения

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



### Оптимизации

Современные движки регулярных выражений могут оптимизировать внутренние алгоритмы ради ускорения. Так что их работа может несколько отличаться от описанного алгоритма.

Но эти внутренние оптимизации для нас незаметны, снаружи всё будет работать, как описано.

Сложные регулярные выражения трудно оптимизировать, так что поиск может работать и в точности так, как было описано.

## Альтернативный подход

С регулярными выражениями часто есть несколько путей добиться одного и того же результата.

В нашем случае мы можем найти кавычки без использования ленивого режима с помощью регулярного выражения `"[^\"]+"`:

```
1 let regexp = /"[^\"]+"/g;
2
3 let str = 'a "witch" and her "broom" is one';
4
5 alert( str.match(regexp) ); // witch, broom
```

Регулярное выражение `"[^\"]+"` получит нужный результат, потому что оно ищет кавычку `'"`, за которой следует один или несколько символов «не-кавычек» `[^\"]`, а затем – закрывающая кавычка.

Движок регулярного выражения набирает, сколько может, `[^\"]+`, пока не встречает закрывающую кавычку, на которой останавливается.

Обратите внимание, что эта логика не заменяет ленивые квантификаторы!

Просто она работает по-другому. Временами нужен один вариант, временами – другой.

**Давайте посмотрим пример, в котором ленивый квантификатор не справляется, а этот вариант работает правильно.**

Например, мы хотим найти ссылки вида `<a href="..." class="doc">`, с произвольным href.

Какое регулярное выражение нам нужно использовать?

Первой мыслью может быть: `/<a href=".*" class="doc">/g`.

Давайте проверим:

```
1 let str = '...<a href="link" class="doc">...';
2 let regexp = /<a href=".*" class="doc">/g;
3
4 // Работает!
5 alert( str.match(regexp) ); // <a href="link" class="do
```

Регулярное выражение работает. Но давайте посмотрим, что произойдёт, если в тексте будет много ссылок?

```
1 let str = '...<a href="link1" class="doc">... <a href="
2 let regexp = /<a href=".*" class="doc">/g;
3
4 // Упс! Две ссылки в одном совпадении!
5 alert( str.match(regexp) ); // <a href="link1" class="d
```

В данном случае мы получили неправильный результат по той же причине, что в примере с «witches». Квантификатор `.*` забирает слишком много символов.

Совпадение будет выглядеть так:

```
1 <a href="....." class="doc">... <a href="link2" class="doc">...
```



Давайте изменим шаблон, сделав квантификатор ленивым .\*?:

```
1 let str = '...<a href="link1" class="doc">... <a href="link2" class="doc">...'
2 let regexp = /<a href=".*?" class="doc">/g;
3
4 // Работает!
5 alert( str.match(regexp) ); // <a href="link1" class="doc">...
```

Теперь кажется, что всё работает правильно. У нас есть два совпадения:

```
1 <a href="....." class="doc">... <a href="....." class="doc">...
2 <a href="link1" class="doc">... <a href="link2" class="doc">...
```

...Но давайте попробуем его на ещё одном тексте:

```
1 let str = '...<a href="link1" class="wrong">... <p style="color: red;">...'
2 let regexp = /<a href=".*?" class="doc">/g;
3
4 // Неправильное совпадение!
5 alert( str.match(regexp) ); // <a href="link1" class="wrong">...
```

Ну вот, ленивый квантификатор нас подвёл. В совпадении находится не только ссылка, но и текст после неё, включая `<p...>`.



Почему?



Происходит следующее:

1. Первым делом регулярное выражение находит начало ссылки `<a href="`.
2. Затем оно ищет `.*`, берёт один символ (лениво!) и проверяет, есть ли совпадение для `" class="doc">` (нет).
3. Затем берёт другой символ для `.*`, и так далее... пока не достигнет `" class="doc">`. Поиск завершён.

Но с этим есть проблема: конец совпадения находится уже за границей ссылки `<a...>`, вообще в другом теге `<p>`. Что нам не подходит.

Вот как совпадение выглядит по отношению к исходному тексту:

```
1 <a href="....." class="doc">... <p style="color: red;">...
2 <a href="link1" class="wrong">... <p style="color: red;">...
```

Итак, нужен шаблон для поиска `<a href="...something..." class="doc">`, но и с ленивым и с жадным режимами есть проблема.

Правильным вариантом может стать: `href="[^\"]*`. Он найдёт все символы внутри атрибута `href` до ближайшей следующей кавычки, как раз то, что нам нужно.

Работающий пример:

```
1 let str1 = '...<a href="link1" class="wrong">... <p style="color: red;">...'
2 let str2 = '...<a href="link1" class="doc">... <a href="link2" class="doc">...'
3 let regexp = /<a href="[^\"]*" class="doc">/g;
4
5 // Работает!
6 alert( str1.match(regexp) ); // совпадений нет, всё правильно
7 alert( str2.match(regexp) ); // <a href="link1" class="doc">...
```

Раздел

Регулярные выражения

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub

Раздел

[Регулярные выражения](#)

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



## Итого

У квантификаторов есть два режима работы:

### Жадный

По умолчанию движок регулярного выражения пытается повторить квантификатор столько раз, сколько это возможно. Например, `\d+` получит все возможные цифры. Когда цифры закончатся или он дойдёт до конца строки, движок продолжит искать совпадение для оставшегося шаблона. Если совпадения не будет, он уменьшит количество повторов (осуществит возврат) и попробует снова.

### Ленивый

Включается с помощью знака вопроса `?` после квантификатора. Движок регулярного выражения пытается найти совпадение для оставшегося шаблона перед каждым повторением квантификатора.

Как мы увидели на примере поиска строк в кавычках, ленивый режим не «панацея» от всех проблем жадного поиска. В качестве альтернативы может выступать «хорошо настроенный» жадный поиск, как в шаблоне `"[^\"]+"`.

## ✓ Задачи

### Совпадение для `/d+? d+?/`

Какое здесь будет совпадение?

```
1 "123 456".match(/d+? d+?/g) ; // ?
```

решение



### Поиск HTML-комментариев

Найти все HTML-комментарии в тексте:

```
1 let regexp = /ваше регулярное выражение/g;
2
3 let str = `... <!-- My -- comment
4 test --> .. <!--> ..
5 `;
6
7 alert( str.match(regexp) ); // '<!-- My -- comment \n t
```

решение



### Поиск HTML-тегов

Создайте регулярное выражение, чтобы найти все (открывающие и закрывающие) HTML-теги с их атрибутами.

Пример использования:

```
1 let regexp = /ваше регулярное выражение/g;
2
3 let str = '<> <a href="/"> <input type="radio" checked>
4
5 alert( str.match(regexp) ); // '<a href="/">', '<input
```

В этой задаче мы предполагаем, что теги выглядят как `<...>` что угодно `...`, и внутри тегов не может быть символов `<` и `>` (первый встреченный `>` закрывает тег).

решение

Раздел

[Регулярные выражения](#)

Навигация по уроку

Жадный поиск

Ленивый режим

Альтернативный подход

Итого

Задачи (3)

Комментарии

Поделиться



Редактировать на GitHub



## Комментарии

перед тем как писать...

© 2007–2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

