Раздел
Регулярные выражения
Комментарии
Поделиться
Редактировать на GitHub

Поиск на заданной позиции, флаг "y"

Флаг `y` позволяет произвести поиск на определённой позиции в исходной строке.

Чтобы разобрать флаг `y` и понять, чем же он хорош, рассмотрим практический пример.

Одна из часто встречающихся задач регулярных выражений – лексический разбор: мы имеем текст, например, на каком-то языке программирования и получаем его структурные элементы.

Например, в HTML есть теги и атрибуты, в JavaScript-коде – переменные и функции, и т.п.

Мы не будем погружаться глубоко в тему написания таких анализаторов (это специализированная область со своим набором инструментов и алгоритмов). Но в процессе их работы, вообще, в процессе анализа текста, очень часто возникает задача «прочитать что-то на заданной позиции».

Например, у нас есть строка кода `let varName = "value"`, и нам надо прочитать из неё имя переменной, которое начинается с позиции `4`.

Имя переменной будем искать как слово `\w+`. Вообще, в языке JavaScript для имени переменной нужно чуть более сложное регулярное выражение, но здесь это не важно.

Вызов `str.match(/\w+/)` найдёт только первое слово в строке или все слова (с флагом `g`), а нам нужно одно слово именно на позиции `4`.

Для поиска, начиная с нужной позиции, можно использовать метод `regexp.exec(str)`.

Если у регулярного выражения `regexp` нет флагов `g` или `y`, то этот метод ищет первое совпадение в строке `str`, точно так же, как `str.match(regexp)`. Здесь нас этот простейший вариант без флагов не интересует.

Если флаг `g` есть, то он осуществляет поиск в строке `str`, начиная с позиции, заданной свойством `regexp.lastIndex`. И, когда находит, обновляет `regexp.lastIndex` на позицию после совпадения.

При создании регулярного выражения его свойство `lastIndex` равно `0`.

Так что повторные вызовы `regexp.exec` возвращают совпадения по очереди, одно за другим.

Например (с флагом `g`):

```
let str = 'let varName';

let regexp = /\w+/g;
alert(regexp.lastIndex); // 0 (при создании lastIndex=0

let word1 = regexp.exec(str);
alert(word1[0]); // let (первое слово)
alert(regexp.lastIndex); // 3 (позиция за первым совпад

let word2 = regexp.exec(str);
alert(word2[0]); // varName (второе слово)
alert(regexp.lastIndex); // 11 (позиция за вторым совпа,

let word3 = regexp.exec(str);
alert(word3); // null (больше совпадений нет)
alert(regexp.lastIndex); // 0 (сбрасывается по окончани
```

Заметим, что каждое совпадение возвращается в виде массива, со всеми скобочными группами и дополнительными свойствами.

Можно перебрать все совпадения в цикле:

```
let str = 'let varName';
let regexp = /\w+/g;

let result;

while (result = regexp.exec(str)) {
  alert( `Найдено ${result[0]} на позиции ${result.inde
  // Найдено let на позиции 0, затем
  // Найдено varName на позиции 4
}
```

Такое использование `regexp.exec` представляет собой альтернативу методу `str.matchAll`.

Таким образом, последовательные вызовы `regexp.exec` могут найти все совпадения, представляя собой альтернативу методам `str.match/matchAll`.

Но, в отличие от других методов, мы можем поставить самостоятельно `lastIndex`, начав тем самым поиск именно с нужной позиции.

Например, найдём слово, начиная с позиции `4`:

```
let str = 'let varName = "value"';

let regexp = /\w+/g; // без флага g свойство lastIndex

regexp.lastIndex = 4;

let word = regexp.exec(str);
alert(word); // varName
```

Поиск `\w+` произведён, начиная с позиции `regexp.lastIndex = 4`.

Заметим, что такой поиск лишь начинается с позиции `lastIndex` и идёт дальше. Если слова на позиции `lastIndex` нет, но оно есть позже, оно всё равно будет найдено:

```
let str = 'let varName = "value"';

let regexp = /\w+/g;

regexp.lastIndex = 3;

let word = regexp.exec(str);
alert(word[0]); // varName
alert(word.index); // 4
```

...То есть, при флаге `g` свойство `lastIndex` задаёт начальную позицию поиска.

Флаг `y` заставляет `regexp.exec` искать ровно на позиции `lastIndex`, ни до и ни после.

Вот тот же поиск с флагом `y`:

```
let str = 'let varName = "value"';

let regexp = /\w+/y;

regexp.lastIndex = 3;
alert( regexp.exec(str) ); // null (на позиции 3 пробел

regexp.lastIndex = 4;
alert( regexp.exec(str) ); // varName (слово на позиции
```

Как можно видеть, регулярное выражение `/\w+/y` не найдено на позиции `3` (в отличие от флага `g`), но найдено на позиции `4`.

Представим себе, что у нас большой текст, и в нём нет ни одного совпадения. В таком случае регулярное выражение с флагом `g` будет идти до самого конца текста, и это займёт гораздо больше времени, чем поиск с флагом `y`.

В задачах, подобных лексическому анализу, обычно много поисков на конкретной позиции. Использование флага `y` – ключ к хорошей производительности.

Комментарии

перед тем как писать...