2-го октября 2020

Методы RegExp и String

В этой главе мы рассмотрим все детали методов для работы с регулярными выражениями.

str.match(regexp)

Метод `str.match(regexp)` ищет совпадения с `regexp` в строке `str`.

У него есть три режима работы:

1. Если у регулярного выражения нет флага `g`, то он возвращает первое совпадение в виде массива со скобочными группами и свойствами `index` (позиция совпадения), `input` (строка поиска, равна `str`):

```
let str = "I love JavaScript";

let result = str.match(/Java(Script)/);

alert( result[0] );     // JavaScript (всё совпадение
alert( result[1] );     // Script (первые скобки)
alert( result.length ); // 2

// Дополнительная информация:
alert( result.index );  // 7 (позиция совпадения)
alert( result.input );  // I love JavaScript (исходна
```

2. Если у регулярного выражения есть флаг `g`, то он возвращает массив всех совпадений, без скобочных групп и других деталей.

```
let str = "I love JavaScript";

let result = str.match(/Java(Script)/g);

alert( result[0] ); // JavaScript
alert( result.length ); // 1
```

3. Если совпадений нет, то, вне зависимости от наличия флага `g`, возвращается `null`.

   Это очень важный нюанс. При отсутствии совпадений возвращается не пустой массив, а именно `null`. Если об этом забыть, можно легко допустить ошибку, например:

```
let str = "I love JavaScript";

let result = str.match(/HTML/);

alert(result); // null
alert(result.length); // Ошибка: у null нет свойства
```

   Если хочется, чтобы результатом всегда был массив, можно написать так:

```
let result = str.match(regexp) || [];
```

str.matchAll(regexp)

> **Новая возможность**
>
> Эта возможность была добавлена в язык недавно. В старых браузерах может понадобиться полифил.

Метод `str.matchAll(regexp)` – «новый, улучшенный» вариант метода `str.match`.

Он используется, в первую очередь, для поиска всех совпадений вместе со скобочными группами.

У него 3 отличия от `match`:

1. Он возвращает не массив, а перебираемый объект с результатами, обычный массив можно сделать при помощи `Array.from`.
2. Каждое совпадение возвращается в виде массива со скобочными группами (как `str.match` без флага `g`).
3. Если совпадений нет, то возвращается не `null`, а пустой перебираемый объект.

Пример использования:

```
let str = '<h1>Hello, world!</h1>';
let regexp = /<(.*?)>/g;

let matchAll = str.matchAll(regexp);

alert(matchAll); // [object RegExp String Iterator], не

matchAll = Array.from(matchAll); // теперь массив

let firstMatch = matchAll[0];
alert( firstMatch[0] );  // <h1>
alert( firstMatch[1] );  // h1
alert( firstMatch.index );  // 0
alert( firstMatch.input );  // <h1>Hello, world!</h1>
```

При переборе результатов `matchAll` в цикле `for..of` вызов `Array.from`, разумеется, не нужен.

str.split(regexp|substr, limit)

Разбивает строку в массив по разделителю – регулярному выражению regexp или подстроке substr.

Обычно мы используем метод `split` со строками, вот так:

```
alert('12-34-56'.split('-')) // массив [12, 34, 56]
```

Но мы можем разделить по регулярному выражению аналогичным образом:

```
alert('12, 34, 56'.split(/,\s*/)) // массив [12, 34, 56
```

str.search(regexp)

Метод `str.search(regexp)` возвращает позицию первого совпадения с `regexp` в строке `str` или `-1`, если совпадения нет.

Например:

```
let str = "Я люблю JavaScript!";

let regexp = /Java.+/;

alert( str.search(regexp) ); // 8
```

Важное ограничение: `str.search` умеет возвращать только позицию первого совпадения.

Если нужны позиции других совпадений, то следует использовать другой метод, например, найти их все при помощи `str.matchAll(regexp)`.

str.replace(str|regexp, str|func)

Это универсальный метод поиска-и-замены, один из самых полезных. Этакий швейцарский армейский нож для поиска и замены в строке.

Мы можем использовать его и без регулярных выражений, для поиска-и-замены подстроки:

```
1  // заменить тире двоеточием
2  alert('12-34-56'.replace("-", ":")) // 12:34-56
```

Хотя есть подводный камень.

Когда первый аргумент `replace` является строкой, он заменяет только первое совпадение.

Вы можете видеть это в приведённом выше примере: только первый `"-"` заменяется на `":"`.

Чтобы найти все дефисы, нам нужно использовать не строку `"-"`, а регулярное выражение `/-/g` с обязательным флагом `g`:

```
1  // заменить все тире двоеточием
2  alert( '12-34-56'.replace( /-/g, ":" ))  // 12:34:56
```

Второй аргумент – строка замены. Мы можем использовать специальные символы в нем:

Спецсимволы	Действие в строке замены
`$$`	вставляет `"$"`
`$&`	вставляет всё найденное совпадение
$`	вставляет часть строки до совпадения
`$'`	вставляет часть строки после совпадения
`$n`	если `n` это 1-2 значное число, то вставляет содержимое n-й скобки
`$<имя>`	вставляет содержимое скобки с указанным именем

Например:

```
1  let str = "John Smith";
2
3  // поменять местами имя и фамилию
4  alert(str.replace(/(\w+) (\w+)/i, '$2, $1')) // Smith,
```

Для ситуаций, которые требуют «умных» замен, вторым аргументом может быть функция.

Она будет вызываться для каждого совпадения, и её результат будет вставлен в качестве замены.

Функция вызывается с аргументами `func(match, p1, p2, ..., pn, offset, input, groups)`:

1. `match` – найденное совпадение,
2. `p1, p2, ..., pn` – содержимое скобок (см. главу Скобочные группы).
3. `offset` – позиция, на которой найдено совпадение,
4. `input` – исходная строка,
5. `groups` – объект с содержимым именованных скобок (см. главу Скобочные группы).

Если скобок в регулярном выражении нет, то будет только 3 аргумента: `func(match, offset, input)`.

Например, переведём выбранные совпадения в верхний регистр:

```
let str = "html and css";

let result = str.replace(/html|css/gi, str => str.toUpp

alert(result); // HTML and CSS
```

Заменим каждое совпадение на его позицию в строке:

```
alert("Xo-Xo-xo".replace(/xo/gi, (match, offset) => off
```

В примере ниже две скобки, поэтому функция замены вызывается с 5-ю аргументами: первый – всё совпадение, затем два аргумента содержимое скобок, затем (в примере не используются) индекс совпадения и исходная строка:

```
let str = "John Smith";

let result = str.replace(/(\w+) (\w+)/, (match, name, s

alert(result); // Smith, John
```

Если в регулярном выражении много скобочных групп, то бывает удобно использовать остаточные аргументы для обращения к ним:

```
let str = "John Smith";

let result = str.replace(/(\w+) (\w+)/, (...match) => `

alert(result); // Smith, John
```

Или, если мы используем именованные группы, то объект `groups` с ними всегда идёт последним, так что можно получить его так:

```
let str = "John Smith";

let result = str.replace(/(?<name>\w+) (?<surname>\w+)/
  let groups = match.pop();

  return `${groups.surname}, ${groups.name}`;
});

alert(result); // Smith, John
```

Использование функции даёт нам максимальные возможности по замене, потому что функция получает всю информацию о совпадении, имеет доступ к внешним переменным и может делать всё что угодно.

regexp.exec(str)

Метод `regexp.exec(str)` ищет совпадение с `regexp` в строке `str`. В отличие от предыдущих методов, вызывается на регулярном выражении, а не на строке.

Он ведёт себя по-разному в зависимости от того, имеет ли регулярное выражение флаг `g`.

Если нет `g`, то `regexp.exec(str)` возвращает первое совпадение в точности как `str.match(regexp)`. Такое поведение не даёт нам ничего нового.

Но если есть `g`, то:

- Вызов `regexp.exec(str)` возвращает первое совпадение и *запоминает* позицию после него в свойстве `regexp.lastIndex`.
- Следующий такой вызов начинает поиск с позиции `regexp.lastIndex`, возвращает следующее совпадение и запоминает позицию после него в `regexp.lastIndex`.
- ...И так далее.
- Если совпадений больше нет, то `regexp.exec` возвращает `null`, а для `regexp.lastIndex` устанавливается значение `0`.

Таким образом, повторные вызовы возвращают одно за другим все совпадения, используя свойство `regexp.lastIndex` для отслеживания текущей позиции поиска.

В прошлом, до появления метода `str.matchAll` в JavaScript, вызов `regexp.exec` в цикле использовали для получения всех совпадений с их позициями и группами скобок в цикле:

```
let str = 'Больше о JavaScript на https://javascript.in
let regexp = /javascript/ig;

let result;

while (result = regexp.exec(str)) {
  alert( `Найдено ${result[0]} на позиции ${result.inde
  // Найдено JavaScript на позиции 9, затем
  // Найдено javascript на позиции 31
}
```

Это работает и сейчас, хотя для современных браузеров `str.matchAll`, как правило, удобнее.

Мы можем использовать `regexp.exec` для поиска совпадения, начиная с нужной позиции, если вручную поставим `lastIndex`.

Например:

```
let str = 'Hello, world!';

let regexp = /\w+/g; // без флага g свойство lastIndex
regexp.lastIndex = 5; // ищем с 5-й позиции (т.е с запя

alert( regexp.exec(str) ); // world
```

Если у регулярного выражения стоит флаг `y`, то поиск будет вестись не начиная с позиции `regexp.lastIndex`, а только на этой позиции (не далее в тексте).

В примере выше заменим флаг `g` на `y`. Ничего найдено не будет, поскольку именно на позиции `5` слова нет:

```
let str = 'Hello, world!';

let regexp = /\w+/y;
regexp.lastIndex = 5; // ищем ровно на 5-й позиции

alert( regexp.exec(str) ); // null
```

Это удобно в тех ситуациях, когда мы хотим «прочитать» что-то из строки по регулярному выражению именно на конкретной позиции, а не где-то далее.

regexp.test(str)

Метод `regexp.test(str)` ищет совпадение и возвращает `true/false`, в зависимости от того, находит ли он его.

Например:

```
let str = "Я люблю JavaScript";

// эти два теста делают одно и же
alert( /люблю/i.test(str) ); // true
alert( str.search(/люблю/i) != -1 ); // true
```

Пример с отрицательным ответом:

```
let str = "Ля-ля-ля";

alert( /люблю/i.test(str) ); // false
alert( str.search(/люблю/i) != -1 ); // false
```

Если регулярное выражение имеет флаг `g`, то `regexp.test` ищет, начиная с `regexp.lastIndex` и обновляет это свойство, аналогично `regexp.exec`.

Таким образом, мы можем использовать его для поиска с заданной позиции:

```
let regexp = /люблю/gi;

let str = "Я люблю JavaScript";

// начать поиск с 10-й позиции:
regexp.lastIndex = 10;
alert( regexp.test(str) ); // false (совпадений нет)
```

Одно и то же регулярное выражение, использованное повторно на другом тексте, может дать другой результат

Если мы применяем одно и то же регулярное выражение последовательно к разным строкам, это может привести к неверному результату, поскольку вызов `regexp.test` обновляет свойство `regexp.lastIndex`, поэтому поиск в новой строке может начаться с ненулевой позиции.

Например, здесь мы дважды вызываем `regexp.test` для одного и того же текста, и второй раз поиск завершается уже неудачно:

```
let regexp = /javascript/g;  // (regexp только что

alert( regexp.test("javascript") ); // true (тепер
alert( regexp.test("javascript") ); // false
```

Это именно потому, что во втором тесте `regexp.lastIndex` не равен нулю.

Чтобы обойти это, можно присвоить `regexp.lastIndex = 0` перед новым поиском. Или вместо методов на регулярном выражении вызывать методы строк `str.match/search/...`, они не используют `lastIndex`.

Комментарии

перед тем как писать...