

Раздел

[Регулярные выражения](#)

Навигация по уроку

Регулярные выражения

Флаги

Поиск: str.match


Замена: str.replace

Проверка: regexp.test

Итого

Комментарии

Поделиться

[Редактировать на GitHub](#) → [Регулярные выражения](#) 8-го сентября 2019

Введение: шаблоны и флаги

Регулярные выражения – мощное средство поиска и замены в строке.

В JavaScript регулярные выражения реализованы отдельным объектом [RegExp](#) и интегрированы в методы строк.

Регулярные выражения

Регулярное выражение (оно же «регэксп», «регулярка» или просто «рег»), состоит из *шаблона* (также говорят «паттерн») и необязательных *флагов*.

Существует два синтаксиса для создания регулярного выражения.

«Длинный» синтаксис:

```
1  regexp = new RegExp("шаблон", "флаги");
```

...И короткий синтаксис, использующий слеш `/` :

```
1  regexp = /шаблон/; // без флагов
2  regexp = /шаблон/gmi; // с флагами gmi (будут описаны д
```

Слеш `/.../` говорят JavaScript о том, что это регулярное выражение. Они играют здесь ту же роль, что и кавычки для обозначения строк.

Регулярное выражение `regexp` в обоих случаях является объектом встроенного класса `RegExp`.

Основная разница между этими двумя способами создания заключается в том, что слеш `/.../` не допускают никаких вставок переменных (наподобие возможных в строках через `${...}`). Они полностью статичны.

Слеш используются, когда мы на момент написания кода точно знаем, каким будет регулярное выражение – и это большинство ситуаций. А `new RegExp` – когда мы хотим создать регулярное выражение «на лету» из динамически сгенерированной строки, например:

```
1  let tag = prompt("Какой тег вы хотите найти?", "h2");
2
3  let regexp = new RegExp(`<${tag}>`); // то же, что /<h2
```

Флаги

Регулярные выражения могут иметь флаги, которые влияют на поиск.

В JavaScript их всего шесть:

i

С этим флагом поиск не зависит от регистра: нет разницы между `A` и `a` (см. пример ниже).

g

С этим флагом поиск ищет все совпадения, без него – только первое.

m

Многострочный режим (рассматривается в главе [Многострочный режим якорей ^ \\$, флаг "m"](#)).

s

Раздел

Регулярные выражения

Навигация по уроку

Регулярные выражения

Флаги

Поиск: `str.match`

Замена: `str.replace`

Проверка: `regex.test`

Итого

Комментарии

Поделиться



Редактировать на GitHub



Включает режим «dotall», при котором точка `.` может соответствовать символу перевода строки `\n` (рассматривается в главе [Символьные классы](#)).

u

Включает полную поддержку юникода. Флаг разрешает корректную обработку суррогатных пар (подробнее об этом в главе [Юникод: флаг "u" и класс \p{...}](#)).

y

Режим поиска на конкретной позиции в тексте (описан в главе [Поиск на заданной позиции, флаг "y"](#))

Цветовые обозначения

Здесь и далее в тексте используется следующая цветовая схема:

- регулярное выражение – красный
- строка (там где происходит поиск) – синий
- результат – зелёный

Поиск: `str.match`

Как уже говорилось, использование регулярных выражений интегрировано в методы строк.

Метод `str.match(regex)` для строки `str` возвращает совпадения с регулярным выражением `regex`.

У него есть три режима работы:

1. Если у регулярного выражения есть флаг `g`, то он возвращает массив всех совпадений:

```
1 let str = "Любо, братцы, любо!";
2
3 alert( str.match(/любо/gi) ); // Любо,любо (массив из
```

Обратите внимание: найдены и Любо и любо, благодаря флагу `i`, который делает регулярное выражение регистронезависимым.

2. Если такого флага нет, то возвращает только первое совпадение в виде массива, в котором по индексу `0` находится совпадение, и есть свойства с дополнительной информацией о нём:

```
1 let str = "Любо, братцы, любо!";
2
3 let result = str.match(/любо/i); // без флага g
4
5 alert( result[0] ); // Любо (первое совпадение)
6 alert( result.length ); // 1
7
8 // Дополнительная информация:
9 alert( result.index ); // 0 (позиция совпадения)
10 alert( result.input ); // Любо, братцы, любо! (исход
```

В этом массиве могут быть и другие индексы, кроме `0`, если часть регулярного выражения выделена в скобки. Мы разберём это в главе [Скобочные группы](#).

3. И, наконец, 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:

Раздел

Регулярные выражения

Навигация по уроку

Регулярные выражения

Флаги

Поиск: str.match

Замена: str.replace

Проверка: regexp.test

Итого

Комментарии

Поделиться



Редактировать на GitHub



```
1 let matches = "JavaScript".match(/HTML/); // = null
2
3 if (!matches.length) { // Ошибка: у null нет свойства
4   alert("Ошибка в строке выше");
5 }
```

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

```
1 let matches = "JavaScript".match(/HTML/) || [];
2
3 if (!matches.length) {
4   alert("Совпадений нет"); // теперь работает
5 }
```

Замена: str.replace

Метод `str.replace(regex, replacement)` заменяет совпадения с `regex` в строке `str` на `replacement` (все, если есть флаг `g`, иначе только первое).

Например:

```
1 // без флага g
2 alert( "We will, we will".replace(/we/i, "I") ); // I w
3
4 // с флагом g
5 alert( "We will, we will".replace(/we/ig, "I") ); // I ,
```

В строке замены `replacement` мы можем использовать специальные комбинации символов для вставки фрагментов совпадения:



Спецсимволы	Действие в строке замены
<code>\$&</code>	вставляет всё найденное совпадение
<code>\$`</code>	вставляет часть строки до совпадения
<code>\$'</code>	вставляет часть строки после совпадения
<code>\$n</code>	если <code>n</code> это 1-2 значное число, вставляет содержимое <code>n</code> -й скобочной группы регулярного выражения, больше об этом в главе Скобочные группы
<code>\$<name></code>	вставляет содержимое скобочной группы с именем <code>name</code> , также изучим в главе Скобочные группы
<code>\$\$</code>	вставляет символ <code>"\$"</code>



Пример с `$&`:

```
1 alert( "Люблю HTML".replace(/HTML/, "$& и JavaScript") );
```

Проверка: regexp.test

Метод `regexp.test(str)` проверяет, есть ли хоть одно совпадение, если да, то возвращает `true`, иначе `false`.

```
1 let str = "Я Люблю JavaScript";
2 let regexp = /люблю/i;
3
4 alert( regexp.test(str) ); // true
```

Далее в этом разделе мы будем изучать регулярные выражения, увидим ещё много примеров их использования, а также познакомимся с другими методами.

Полная информация о различных методах дана в главе [Методы RegExp и String](#).

Итого

Раздел

[Регулярные выражения](#)

Навигация по уроку

Регулярные выражения

Флаги

Поиск: `str.match`

Замена: `str.replace`

Проверка: `regex.test`

Итого

Комментарии

Поделиться



Редактировать на GitHub



- Регулярное выражение состоит из шаблона и необязательных флагов: `g`, `i`, `m`, `u`, `s`, `y`.
- Без флагов и специальных символов, которые мы изучим позже, поиск по регулярному выражению аналогичен поиску подстроки.
- Метод `str.match(regex)` ищет совпадения: все, если есть флаг `g`, иначе только первое.
- Метод `str.replace(regex, replacement)` заменяет совпадения с `regex` на `replacement`: все, если у регулярного выражения есть флаг `g`, иначе только первое.
- Метод `regex.test(str)` возвращает `true`, если есть хоть одно совпадение, иначе `false`.

Проводим [курсы по JavaScript и фреймворкам](#). ✕

Комментарии

перед тем как писать...

