**РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ**

Регулярные выражения (Regular expression) — механизм, который позволяет с помощью специального шаблона осуществлять поиск определенного фрагмента текста (подстроки) в заданной строке текста.

С помощью регулярного выражения вы можете узнать:

* • встречается ли какое-то конкретное слово или строка в фрагменте текста (true/false);
* • осуществить замену искомого текста на другой фрагмент текста;
* • найти не конкретную строку, а строку которая соответствует определенному шаблону (например: первый символ — цифра,  
  длина 6 символов, последний символ — точка).

Синтаксис:

/pattern/flag

// — указывают на начало и конец регулярного выражения, пободно кавычкам для обозначения строк.

pattern — шаблон искомого фрагмента строки, заданный с использованием специального синтаксиса.  
flag — специальный модификатор, который позволяет более точно настроить способ поиска.

Значения флагов (модификаторов):

* • i — производить поиск независимо от регистра символов;
* • g — искать все совпадения символов во всей строке (если флага нет — ищем только первое совпадение);
* • m — многострочный режим, используется если строка, в которой осуществляется поиск, содержит переносы строк.

Некоторые примеры регулярных выражений:

/test/gi //поиск строки test не зависимо от регистра

/test/g //поиск строки test

/^ t.t$/gi //поиск строки с первым и последним символом t и произвольным буквенным символом между ними

/\d{2-3}/i //поиск числа с указанием количества повторений

/[0-9]/ //поиск символа числа в диапозоне от 0 до 9

/\d+/ig //поиск числа, которое может повоториться более 1 раза

Создание регулярного выражения

В JavaScript регулярные выражения можно создавать двумя способами:

* • используя литерал регулярного выражения /exp/
* • вызвав функцию конструктор RegExp

Литерал регулярного выражения

let exp = /start/i;

Этот подход обычно используется если вы заранее знаете, что ваше выражение не будет меняться.  
Такое выражение записывается один раз, компилируется и затем используется в неизменном виде.

let str = 'Time to start';

let exp = /^s.{3}t$/i;

exp.test(str);

Вызов функции конструктора RegExp

let exp = new RegExp("start", "i");

Этот подход часто используется если шаблон заранее не известен. То есть может каким то образом вычисляться либо передаваться как параметр.

function findText(pattern){

let str = 'Time to start';

let exp = new RegExp(pattern, "i");

return exp.test(str);

}

findText("^s.{3}t$");

findText("Time");

Методы для работы с регулярными выражениями

Некоторые методы для работы со строками поддерживают использование регулярных выражений:

* str.search(reg)
* str.match (reg)
* str.replace(reg, peplacement)

Метод search — возвращает позицию искомого элемента в строке или -1 если ничего не найдено.  
Метод способен искать только первое совпадение.  
Используется если необходимо узнать где именно в тексте встречается искомая фраза.

let str = "Time to start";

let exp = /to/g;

console.log(str.search(exp)); // 5

Метод match — возвращает массив совпадений и некоторую дополнительную информацию.  
Если совпадения не были найдены возвращает null.  
Механизм работы отличается в зависимости от использования флага ‘g’

let pattern = new RegExp('t.m', 'i');

let str = 'My name is Tom Cruise';

let result = str.match(pattern);

console.log(result); //["Tom ", index: 11, input: "My name is Tom Cruise"]

let str = 'My name is Tom Cruise. What is your name?';

let result = str.match(/name/g);

console.log(result); //["name", "name"]

Метод replace — осуществляет поиск строки или регулярного выражения с последующей заменой на другую строку.

let str = 'My name is Tom Cruise';

let result = str.replace(/t.m/i, 'Bruce');

console.log(result);

let pattern = /t.m/i;

let str = 'My name is Tom Cruise';

let result = str.replace(pattern, 'Bruce');

console.log(result);

let pattern = new RegExp("t.m", "i");

let str = 'My name is Tom Cruise';

let result = str.replace(pattern, 'Bruce');

console.log(result);

let str = 'My name is Tom Cruise';

let result = str.replace('name', 'Bruce');

console.log(result);

Для множественного поиска необходимо использовать регулярное выражение с флагом ‘g’

let str = 'My name is Tom Cruise. What is your name?';

let result = str.replace(/name/g, 'second name');

console.log(result);

Так же в этом методе дополнительно реализован механизм, который позволяет производить не только простую замену но и более сложную модификацию строк.

$& — вставить в результат всё найденное совпадение.  
$` – вставить в результат часть строки до совпадения.  
$’ – вставить в результат часть строки после совпадения.  
$n – вставить в результат n-ое по счёту совпадение.

Рассмотрим несколько примеров.

let str = 'My name is Tom Cruise';

let result = str.replace(/t.m/i, "Bruce,= $'= ");

console.log(result);//My name is Bruce,= Cruise= Cruise

let str = 'My name is Tom Cruise';

let result = str.replace(/(t.m)/i, "Bruce = $1= ");

console.log(result). //My name is Bruce =Tom= Cruise;

Прототип функции конструктора RegExp так же содержит несколько методов для работы с регулярными выражениями

regexp.test(str);

regexp.exec(str);

Метод test — проверяет есть ли хоть одно совпадение в искомой строке с регулярным выражением. Возвращает true или false.  
Похож на метод str.search().

let str = "Time to start";

let exp = /to/g;

console.log(exp.test(str)); // true

Метод exec — возвращает массив совпадений и дополнительную информацию.  
Если совпадения не были найдены возвращает null.  
Механизм работы отличается в зависимости от использования флага ‘g’  
Похож на метод str.match().

let str = "My name is Tom Cruise. What is your name?";

let exp = /is/;

console.log(exp.exec(str));

//["is", index: 8, input: "My name is Tom Cruise. What is your name?"]

Если использовать флаг ‘g’, модифицирует внутри регулярного выражения свойство lastIndex.  
Используется для поиска в цикле.

let str = "My name is Tom. What is your name?";

let exp = /is/g;

let find;

while ((find = exp.exec(str)) !== null) {

let msg = 'Нашли = >' + find[0] +

'. Продолжим поиск с индекса ' + exp.lastIndex;

console.log(find);

}

//Нашли =>is. Продолжим поиск с индекса 10

//Нашли =>is. Продолжим поиск с индекса 23

Синтаксис

Регулярные выражения поддерживают большое количество специалных символов, который значительно расширяют синтаксис и позволяют задавать практически любые шаблоны.

Якоря

|  |  |  |
| --- | --- | --- |
| Символ | Описание | Пример |
| ^ | Искомое выражение должно находиться в начале текста | /^h/i.test(«hello») |
| $ | Искомое выражение должнонаходиться в конце текста | /o$/i.test(«hello») |
| . | Соответствуют любому одному символу | /h…o/i.test(«hello») /^h.l/i.test(«hello») /^h.l$/i.test(«hello») |
| \b | Соответсввует границам слова | /\bnot\b/i.test(«not me») |
| \B | Не соответсввует границам слова | /\Bis\B/i.test(«Name is Cruise») |

Классы символов

|  |  |  |
| --- | --- | --- |
| Символ | Описание | Пример |
| \s | Пробел и пробельные символы | /is\s/i.test(«Name is Tom») /is\s\w/i.test(«Name is Tom») |
| \S | Не пробел | /\Sis\S/i.test(«Cruise») |
| \d | Цифра Эквивалентно диапазону [0-9] | /\d\sfloor/i.test(«2 floor») /\d\sf/i.test(«2 floor») |
| \D | Не цифра Эквивалентно диапазону [^0-9] | /\D\sfloor/i.test(«second floor») |
| \w | Буква (включая цифры и знак \_) | /.\w./i.test(«[2]») |
| \W | Не буква | /.\W./i.test(«[&]») |

Квантификаторы

|  |  |  |
| --- | --- | --- |
| Символ | Описание | Пример |
| \* | Предыдущий символ повторяется 0 или больше раз | /is\*\s/i.test(«Cruise») |
| + | Предыдущий символ повторяется 1 или больше раз | /cruiz+e/i.test(«Cruise») /cruiz+e/i.test(«Cruizzze») |
| ? | Предыдущий символ повторяется или 1 раз | /cruiz?e/i.test(«Cruise») /cruiz?e/i.test(«Cruizzze») // false |
| {n} | Предыдущий символ повторяется определенное количество раз | /cruis{3}e/i.test(«Cruissse») /^c.{4}e$/i.test(«Cruise») |
| {n,} | Предыдущий символ повторяется как минимум n раз | /cruis{2,}e/i.test(«Cruissse») |
| {n,m} | Предыдущий символ повторяется от n до m раз | /cruis{1,3}e/i.test(«Cruissse») |

Группа и диапазон

|  |  |  |
| --- | --- | --- |
| Символ | Описание | Пример |
| | | Совпадает или символ слева или символ справа | /is\s(Smith|John)/i.test(«My name is Smith») /is\s(Smith|John)/i.test(«My name is Alex») |
| () | Найти выражение в скобках и запомнить это выражение | /(is){2}/i.exec(«My name isis Cruise») /(is){2}/i.exec(«My name is Cruise») //false |
| [] | Входит в диапазон | /[0-9]{3}/i.test(«ab123cdef») |
| [^] | Не входит в диапазон | /[^a-z]{3}/i.test(«1234aaa5678») /[^a-z]{3}/i.test(«1234aa5678») |
| x(?=y) | Соответствует ‘x’ только если за ‘x’ следует ‘y’. | /John\s(?=Smith|Alex)/i.test(«John Smith») /John\s(?=Smith|Alex)/i.test(«John Dou») //false |
| x(?!y) | Соответствует ‘x’ только если за ‘x’ НЕ следует ‘y’. | /John\s(?!Smith|Alex)/i.test(«John Dou») |

Примечание:  
Если необходимо найти в тексте какой-то символ, который непосредственно является спецсимволом, его необходимо предварительно заэкранировать с использованием слеша.

/Joh\^n/i.test("Joh^n");