Регулярные выражения

Знакомство с концепцией регулярных выражений. Регулярные выражения в JavaScript.

[Регулярные выражения](#_2et92p0)

[Методы регулярных выражений в JavaScript](#_tyjcwt)

[Группировки совпадений](#_3dy6vkm)

[Границы поиска и варианты шаблонов](#_1t3h5sf)

[Замена подстрок при помощи PCRE](#_4d34og8)

[Динамические шаблоны](#_2s8eyo1)

[Другие методы поиска](#_17dp8vu)

[Итоги](#_3rdcrjn)

[Домашнее задание](#_qwmts1n4aq8)

[Используемая литература](#_wtjnpeowu7yg)

Во многих задачах программирования рано или поздно появляется проблема недостаточности строгого сравнения данных. Что, если необходимо найти не просто четкое совпадение, но вхождение строки в подстроку или построение того или иного строкового значения по определенному шаблону? Можно, конечно, написать свою логику, но это будет сродни изобретению велосипеда, т.к. данную задачу во многих языках давно решают регулярные выражения.

# Регулярные выражения

**Регулярные выражения, Regular expressions, RegExp, PCRE** – все это синонимы, обозначающие особый способ задания шаблонов текстовых данных. Они порождают свой микроязык программирования, входящий не только в JavaScript, но и во многие другие языки. Особенно удобно, что везде они реализованы практически одинаково.

PCRE очень полезны, но крайне непрезентабельно выглядят. С первого взгляда может показаться, что это просто какой-то случайный набор символов. Однако знание построения PCRE помогает читать их и одновременно неплохо повышает вашу стоимость на рынке.

Как и многое другое в JS, регулярные выражения представляют собой объект. Создать его можно двумя способами: синтаксически и через конструктор.

|  |
| --- |
| var pcre1 = new RegExp("abc"); |
| var pcre2 = /abc/; |

*Заметьте, что во втором примере кавычки ставить не нужно.*

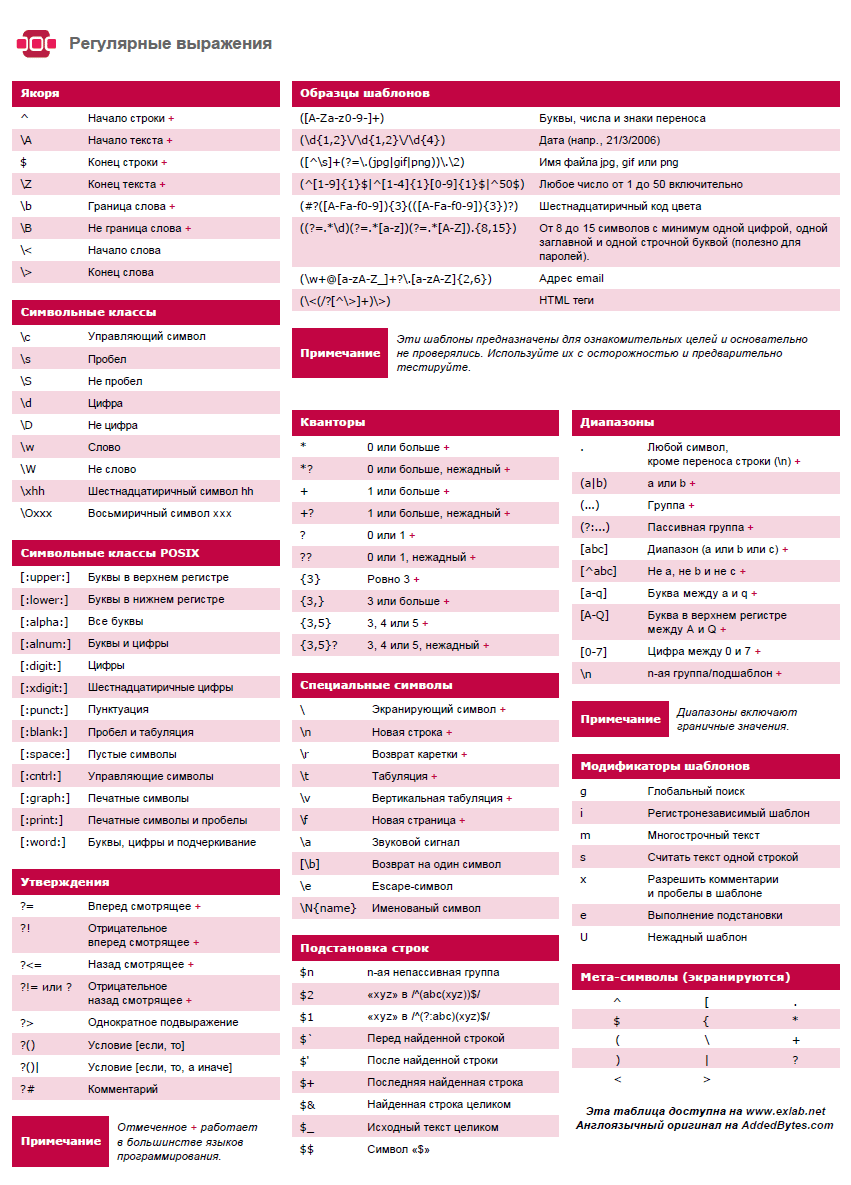
Оба шаблона из примеров говорят, что они умеют искать строгую последовательность символов a, b и c.

При использовании конструктора RegExp шаблон является простой строкой – соответственно, не стоит забывать про экранирующие обратные слеши. Синтаксическая запись, как видим, уже обрамляет шаблон слешами, поэтому они ведут себя здесь иначе.

Шаблон завершается прямым слешем, что говорит о необходимости постановки обратного слеша перед любым прямым слешем внутри шаблона. Точно так же обратные слеши, которые не являются частью специальных символов типа \r или \n, будут сохранены, изменяя таким образом смысл шаблона. У символов типа “?” или “!” в PCRE имеется особое значение, так что при необходимости включения в шаблон именно символа, например вопросительного знака, его обязательно нужно экранировать обратным слешем.

|  |
| --- |
| var pcre3 = /OK\?/; |

Символов, которые применяются в PCRE как специальные, очень много. Их надо учить на практике, что требует времени. Поэтому возьмите за правило на первых порах экранировать любой символ, который не является буквой, цифрой или пробелом.



# Методы регулярных выражений в JavaScript

Будучи объектами, PCRE имеют встроенные методы. Самый простой из них – метод test, который возвращает true, если ему удалось найти шаблонное вхождение в строке, и false в ином случае.

|  |
| --- |
| var pcre4 = /car/; alert(pcre4.test("My car is Bentley")); *// true* var pcre5 = /snatch/; alert(pcre5.test("Harry catches snitch")); *// false* |

Регулярное выражение, которое содержит только буквы, шаблонизирует простую последовательность этих букв, но не всегда всё так просто. PCRE дает возможность работать с гораздо более сложными шаблонами.

Нарастим сложность задачи и скажем, что нам надо найти все номера в тексте. Для этого нужно применить один из двух следующих примеров:

|  |
| --- |
| console.log(/[0123456789]/.test("Olympic games of 2014 were in Sochi")); *// true* console.log(/[0-9]/.test("The answer is 42")); *// true* |

Когда в PCRE набор символов помещен в квадратные скобки, это говорит о том, что часть строки должна совпадать с любым символом этой последовательности. Можно указать тире между двумя символами, тогда интерпретатор будет искать по диапазону в заданных границах в кодировке Unicode. Например, цифры от 0 до 9 в Unicode хранятся подряд, поэтому запись [0-9] захватывает ровно числа от 0 до 9, что на выходе дает совпадение с любым числом.

Можно использовать следующие сокращения для диапазонов:

* \d означает любую цифру.
* \w – любой алфавитно-цифровой символ.
* \s означает пробел, табуляцию, перевод строки и т.п.
* \D – любой символ, кроме цифры.
* \W – любой символ, кроме алфавитно-цифрового.
* \S – любой символ, кроме пробельного.
* . – любой символ, кроме перевода строки.

Используя эту комбинацию символов, можно, например, проверять формат ввода телефонного номера: +7-000-000-00-00

|  |
| --- |
| var phone = /\+\d-\d\d\d-\d\d\d-\d\d-\d\d/; console.log(phone.test("+7-495-000-00-00")); *// true* console.log(phone.test("84950000000")); *// false* |

Это слишком громоздкая запись для и без того малопонятного языка, однако ее можно упростить. Например, выражение [\d.] будет означать любую цифру или точку. Важно заметить, что точка внутри квадратных скобок уже не имеет особого значения и становится просто точкой. Такое же правило действует и для других специальных символов, например «?».

Также есть возможность инверсии набора символов. Ею мы можем указать, какие символы не должны встречаться в искомой строке. Это делается при помощи символа ^

|  |
| --- |
| var pcre6 = /[^01]/; console.log(pcre6.test("110010020100110")); *// недвоичное число* |

Согласитесь, искать символы по одному очень неудобно. Возникает вопрос: «А если надо найти число целиком?». Вот ответ:

|  |
| --- |
| console.log(/'\d+'/.test("2014")); *// true* |

Знак +, поставленный вне квадратных скобок после любой инструкции, означает, что элемент может быть повторен более 1 раза. Если вместо «+» поставить символ \* (звёздочка, asterisk), то шаблон расширится – говорят, что шаблон может встречаться 0 и более раз.

Схожим образом работает символ «?». Он говорит, что часть шаблона объявлена необязательной – она может встретиться, а может и нет.

|  |
| --- |
| var pcre7 = /colou?r/ console.log(pcre7.test("color")); *// true* console.log(pcre7.test("colour")); *// true* |

PCRE также позволяет задать точное количество вхождений шаблона в строке. Для этого используется конструкция {N}, где N – точное число повторений. Наш предыдущий пример с телефоном можно переписать следующим образом:

|  |
| --- |
| var phone = /\+\d-\d{3}-\d{3}-\d{2}-\d{2}/; console.log(phone.test("+7-495-000-00-00")); *// true* console.log(phone.test("84950000000")); *// false* |

Допускается использование открытых интервалов, т.е. запись {,3} будет означать интервал от 0 до 3, а {3,} – 3 и более раз.

Выражения в PCRE можно группировать, применяя круглые скобки. Если заключить в них несколько частей регулярного выражения, они будут считаться одним элементом:

|  |
| --- |
| var cartoonCrying = /boo+(hoo+)+/i; console.log(cartoonCrying.test("Boohoooohoohooo")); *// true* |

Обратите внимание на третий символ «+» – он говорит, что блок hoo+ может встретиться несколько раз. При этом в самом блоке hoo+ последняя буква o может повторяться от 1 раза. Буква i в конце регулярного выражения делает его независимым от регистра, поэтому символ B не отличается от b.

## Группировки совпадений

Выше мы рассматривали все сравнения только при помощи метода test. Он довольно простой и позволяет проверять, есть ли вхождение в строке. Но в JavaScript PCRE имеют более функциональный метод exec, который вернёт объект с данными при нахождении совпадений по шаблону и null в противном случае.

|  |
| --- |
| var pcre8 = /\d+/.exec("Here's the number 100"); console.log(pcre8); *// ["100"]* console.log(pcre8.index); *// 18* |

Как видите, возвращаемый объект match содержит массив найденных строк, а параметр index содержит номер символа, на котором было обнаружено совпадение. Если в регулярном выражении находятся вложенные подвыражения, стоящие в круглых скобках, то текст, подходящий под условия групп, также появится в массиве. Первым элементом будет полное совпадение, вторым – совпадение с 1 группой, и так далее.

|  |
| --- |
| var pcre9 = /'([^']\*)'/; console.log(pcre9.exec("What is 'love'?")); *// ["'love'", "love"]* |

Если группа не найдена в принципе, то в match будет храниться undefined.

Группы отлично подходят для отыскания частей строк. Когда нам нужно не только проверить, хранится ли в строке дата, но и извлечь её, создав нужный объект, мы заключаем последовательности цифр в круглые скобки и находим дату в возвращаемом exec объекте.

## Границы поиска и варианты шаблонов

Зачастую нужно искать от начала строки или до её конца. Для этого применяются символы ^ и $

* ^ говорит, что нужно искать от начала строки.
* $ говорит, что поиск идет до конца строки.

Таким образом, выражение /^\D+$/ будет искать строку, которая состоит только из нецифровых символов.

Если же ограничения на всю строку слишком нестрогие, а нам надо искать только от начала до конца слова, применяется метка \b. В PCRE границей слова считается начало или конец строки, пробел или любое место, где с одной стороны есть символ из группы \w, а с другой – \W.

|  |
| --- |
| console.log(/cat/.test("concatenate")); *// true* console.log(/\bcat\b/.test("concatenate")); *// false* |

Когда мы ищем не строгое совпадение, а вхождение на определенном месте одного из шаблонов (например, нас устроит Audi RS5, Audi RS7 или Audi TTS), можно выполнить следующую инструкцию:

|  |
| --- |
| var pcre10 = /^Audi (RS5|RS7|TTS)s?/; console.log(pcre10.test("Audi RS7")); *// true* console.log(pcre10.test("Audi Q5")); *// false* |

Скобки выделяют часть шаблона, к которой интерпретатор применяет символ | (перечисление). Есть возможность указать несколько подобных операторов друг за другом, тем самым обозначая выбор из 2 и более вариантов.

## Замена подстрок при помощи PCRE

В JavaScript у строковых объектов есть метод replace, заменяющий одну часть строки некой заданной строкой. Для подобной операции вполне можно использовать регулярные выражения.

|  |
| --- |
| console.log("Borobudur".replace(/[ou]/, "a")); *// Barobudur* |

При этом можно использовать модификатор g в конце регулярного выражения, который даст директиву на поиск не первого вхождения, глобальный поиск.

|  |
| --- |
| console.log("Borobudur".replace(/[ou]/g, "a")); *// Barabadar* |

Особенной мощью обладает функционал ссылок на найденные совпадения. К примеру, когда мы работаем со строкой, в которой содержатся ФИО разных людей в формате «Фамилия, Имя», может возникнуть потребность превратить формат в «Имя, Фамилия». Нам поможет следующее выражение:

|  |
| --- |
| console.log("Doe, John\nIvanov, Ivan").replace(/([\w ]+), ([\w ]+)/g, "$2 $1")); |

Конструкции $1 и $2 означают (ссылаются на) группы символов регулярного выражения, находящиеся в скобках. Таким образом, $1 заменяется текстом, который совпал с группой в скобках номер 1, а $2 – номер 2. Максимально – до 9 ссылок. Целое совпадение хранится в ссылке $&.

В качестве аргумента можно передать и функцию. Тогда при каждой замене эта функция будет вызываться, а ее аргументами будут найденные группы.

|  |
| --- |
| var my\_string = "vag vs jdm"; console.log(string.replace(/\b(vag|jdm)\b/g, function(str) {  return str.toUpperCase(); })); *// VAG vs JDM* |

## Динамические шаблоны

Случаются ситуации, когда точный шаблон регулярного выражения неизвестен на момент написания кода. Например, при использовании динамического поиска на сайте вам нужно выделять то слово, которое ввел пользователь. Тогда можно строить строку прямо на лету.

|  |
| --- |
| var search = "функция"; var text = " Тогда при каждой замене эта функция будет вызываться, а её аргументами будут найденные группы"; var pcre11 = new RegExp("(" + search + ")", "gi"); console.log(text.replace(pcre11, "<b>$1</b>")); |

Однако в таком случае обязательно надо рассмотреть ситуацию, когда пользователь может ввести в поисковую строку спецсимволы, которые испортят наше регулярное выражение.

Как мы говорили в самом начале, требуется добавлять обратные слеши перед каждым символом, который мы считаем опасным, но также мы должны ставить обратные слеши перед буквами, т.к. \b или \n – спецсимволы. Тогда можно подготовить нашу поисковую строку, а уже затем вставлять ее в регулярное выражение.

|  |
| --- |
| var name = "dea+hl[]rd"; var text = "Этот dea+hl[]rd всех достал."; var escaped = name.replace(/[^\w\s]/g, "\\$&"); var regexp = new RegExp("\\b(" + escaped + ")\\b", "gi"); console.log(text.replace(regexp, "\_$1\_")); *// Этот \_dea+hl[]rd\_ всех достал.* |

## Другие методы поиска

В JavaScript PCRE есть метод search и свойство lastIndex. Метод принимает на вход регулярное выражение и возвращает индекс первого вхождения совпадения.

|  |
| --- |
| console.log(" word".search(/\S/)); *// 2* console.log(" ".search(/\S/)); *// -1* |

Свойство дает возможность искать вхождения с заданной позиции:

|  |
| --- |
| var pcre11 = /y/g; pcre11.lastIndex = 3; var match = pcre11.exec("xyzzy"); console.log(match.index); *// 4* console.log(pattern.lastIndex); *// 5* |

При успешном поиске вызов exec обновит свойство lastIndex, и оно будет указывать на позицию сразу после найденного вхождения. Если ничего найдено не было, свойство примет значение 0.

# Итоги

Регулярные выражения в JavaScript – это особые объекты, которые представляют собой шаблоны поиска вхождений в строках. Они имеют свой уникальный синтаксис для формирования этих шаблонов.

/abc/ Последовательность символов.

/[abc]/ Любой символ из списка.

/[^abc]/ Любой символ, кроме символов из списка.

/[0-9]/ Любой символ из промежутка.

/x+/ Одно или более вхождений шаблона x.

/x+?/ Одно или более вхождений, нежадное.

/x\*/ Ноль или более вхождений.

/x?/ Ноль или одно вхождение.

/x{2,4}/ От двух до четырех вхождений.

/(abc)/ Группа.

/a|b|c/ Любой из нескольких шаблонов.

/\d/ Любая цифра.

/\w/ Любой алфавитно-цифровой символ («буква»).

/\s/ Любой пробельный символ.

/./ Любой символ, кроме переводов строки.

/\b/ Граница слова.

/^/ Начало строки.

/$/ Конец строки.

Крайне полезно изучение поведения PCRE в онлайн-режиме. Это можно сделать на очень удобном сервисе debuggex.com. Он предоставляет визуализацию регулярного выражения и сравнивает результат с ожиданиями.

# Домашнее задание

1. У вас есть большой текст, в котором для обозначения диалогов используются одинарные кавычки. Придумать шаблон, который меняет одинарные кавычки на двойные.
2. Улучшить шаблон таким образом, чтобы конструкции типа aren’t не меняли одинарную кавычку на двойную.
3. Создать форму обратной связи с полями: Имя, Телефон, e-mail, текст, кнопка «Отправить». При нажатии на кнопку «Отправить» произвести валидацию полей следующим образом:
   1. Имя содержит только буквы.
   2. Телефон подчиняется шаблону +7(000)000-0000.
   3. E-mail выглядит как mymail@mail.ru, или my.mail@mail.ru, или my-mail@mail.ru.
   4. Текст произвольный.
   5. В случае непрохождения валидации одним из полей необходимо выделять это поле красной рамкой и сообщать пользователю об ошибке.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [http://www.pcre.org/.](http://www.pcre.org/)
2. [https://learn.javascript.ru/.](https://learn.javascript.ru/)