Урок 3

Регулярные выражения.

Знакомство с концепцией регулярных выражений. Регулярные выражения в JavaScript.

Регулярные выражения

Методы регулярных выражений в JavaScript

- Группировки совпадений
- Границы поиска и варианты шаблонов
- Замена подстрок при помощи PCRE
- Динамические шаблоны
- Другие методы поиска

Итоги

Домашнее задание

Используемая литература

© geekbrains.ru

Во многих задачах программирования рано или поздно появляется проблема недостаточности строгого сравнения данных. Что, если вам необходимо найти не просто чёткое совпадение, но вхождение строки в подстроку или же построение того или иного строкового значения по определённому 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 – всё это синонимы, обозначают особый способ задания шаблонов текстовых данных. Они порождают свой микроязык программирования, входящий не только в JavaScript, но и многие другие языки программирования. Что особенно радует - 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 искать они умеют строгую последовательность символов a, b и c.

При использовании конструктора RegExp шаблон является простой строкой, соответственно, не стоит забывать про экранирующие обратные слэши. Синтаксическая запись, как видите, уже обрамляет шаблон слэшами, поэтому обратные слэши ведут себя здесь иначе.

Шаблон завершается прямым слэшем, что говорит о необходимости постановки обратного слэша перед любым прямым слешем внутри шаблона. Также обратные слэши, которые не являются частью специальных символов типа \r или \n, будут сохранены, изменяя таким образом смысл шаблона. Также в PCRE у символов типа “?” или “!” 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. И 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.

Регулярные выражения

Якоря

^	Начало строки +
\A	Начало текста +
$	Конец строки +
\Z	Конец текста +
\b	Граница слова +
\B	Не граница слова +
\<	Начало слова
\>	Конец слова

Символьные классы

\c	Управляющий символ
\s	Пробел
\S	Не пробел
\d	Цифра
\D	Не цифра
\w	Слово
\W	Не слово
\xhh	Шестнадцатиричный символ hh
\Oxxx	Восьмиричный символ xxx

Символьные классы POSIX

[:upper:]	Буквы в верхнем регистре
[:lower:]	Буквы в нижнем регистре
[:alpha:]	Все буквы
[:alnum:]	Буквы и цифры
[:digit:]	Цифры
[:xdigit:]	Шестнадцатиричные цифры
[:punct:]	Пунктуация
[:blank:]	Пробел и табуляция
[:space:]	Пустые символы
[:cntrl:]	Управляющие символы
[:graph:]	Печатные символы
[:print:]	Печатные символы и пробелы
[:word:]	Буквы, цифры и подчеркивание

Утверждения

?=	Вперед смотрящее +
?!	Отрицательное вперед смотрящее +
?<=	Назад смотрящее +
?!= или ?	Отрицательное назад смотрящее +
?>	Однократное подвыражение
?()	Условие [если, то]
?()|	Условие [если, то, а иначе]
?#	Комментарий

Примечание *Отмеченное + работает в большинстве языков программирования.*

Образцы шаблонов

([A-Za-z0-9-]+)	Буквы, числа и знаки переноса
(\d{1,2}\/\d{1,2}\/\d{4})	Дата (напр., 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	Имя файла jpg, gif или png
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Любое число от 1 до 50 включительно
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Шестнадцатиричный код цвета
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	От 8 до 15 символов с минимум одной цифрой, одной заглавной и одной строчной буквой (полезно для паролей).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Адрес email
(\<(/?[^\>]+)\>)	HTML теги

Примечание *Эти шаблоны предназначены для ознакомительных целей и основательно не проверялись. Используйте их с осторожностью и предварительно тестируйте.*

Кванторы

*	0 или больше +
*?	0 или больше, нежадный +
+	1 или больше +
+?	1 или больше, нежадный +
?	0 или 1 +
??	0 или 1, нежадный +
{3}	Ровно 3 +
{3,}	3 или больше +
{3,5}	3, 4 или 5 +
{3,5}?	3, 4 или 5, нежадный +

Специальные символы

\	Экранирующий символ +
\n	Новая строка +
\r	Возврат каретки +
\t	Табуляция +
\v	Вертикальная табуляция +
\f	Новая страница +
\a	Звуковой сигнал
[\b]	Возврат на один символ
\e	Escape-символ
\N{name}	Именованый символ

Подстановка строк

$n	n-ая непассивная группа
$2	«xyz» в /^(abc(xyz))$/
$1	«xyz» в /^(?:abc)(xyz)$/
$`	Перед найденной строкой
$'	После найденной строки
$+	Последняя найденная строка
$&	Найденная строка целиком
$_	Исходный текст целиком
$$	Символ «$»

Диапазоны

.	Любой символ, кроме переноса строки (\n) +
(a|b)	a или b +
(...)	Группа +
(?:...)	Пассивная группа +
[abc]	Диапазон (a или b или c) +
[^abc]	Не a, не b и не c +
[a-q]	Буква между a и q +
[A-Q]	Буква в верхнем регистре между A и Q +
[0-7]	Цифра между 0 и 7 +
\n	n-ая группа/подшаблон +

Примечание *Диапазоны включают граничные значения.*

Модификаторы шаблонов

g	Глобальный поиск
i	Регистронезависимый шаблон
m	Многострочный текст
s	Считать текст одной строкой
x	Разрешить комментарии и пробелы в шаблоне
e	Выполнение подстановки
U	Нежадный шаблон

Мета-символы (экранируются)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Эта таблица доступна на www.exlab.net
Англоязычный оригинал на AddedBytes.com

© geekbrains.ru

Методы регулярных выражений в JavaScript

Разумеется, будучи объектами, PCRE имеют встроенные методы. Самый простой из них – это 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(pcre4.test("Harry catches snitch")); // false
```

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

Нарастим сложность задачи и скажем, что нам надо найти все номера в тексте. Для этого нам надо 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 помещён 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, а потому запись [0-9] захватывает ровно числа от 0 до 9, что на выходе даёт совпадение с любым числом.

Также можно использовать следующие сокращения для диапазонов:

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

Но это слишком громоздкая запись для и без того малопонятного языка. Однако, её можно упростить. Например, выражение [\d.] будет означать означает 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, что искать символы по одному очень неудобно. Сначала возникнет 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, т.е. 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, то 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 выражения делает наше регулярное выражение 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 нам просто 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(match);          // ["100"]
console.log(match.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, находящиеся 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. В случае, 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 нам нужно искать от начала строки или до её конца. Для этого применяются символы ^ и $

- ^ говорит, что нужно искать от начала строки.
- $ говорит, что поиск идёт 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, то нужно применять метку \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
```

В случае, когда мы ищем не строгое совпадение, а вхождение на определённом месте одного из пол шаблонов (например, нас устроит Audi RS5, Audi RS7 или Audi TTS), то 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. При этом, мы вполне можем использовать регулярные выражения для подобной операции.

```
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, глобального поиска.

```
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». В этом 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 её 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 ввёл 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, при которой пользователь может ввести в поисковую строку спецсимволы, которые испортят наше регулярное выражение.

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

Также в JavaScript PCRE есть два прекрасных момента: 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 даёт 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 же ничего найдено не было, то 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 четырёх 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

Зачастую крайне полезным будет 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:
    a. Имя содержит только буквы;
    b. Телефон подчиняется шаблону +7(000)000-0000;
    c. E-mail выглядит как mymail@mail.ru, или my.mail@mail.ru, или my-mail@mail.ru
    d. Текст произвольный;
    e. В случае не прохождения 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/
2. https://learn.javascript.ru/