## Регулярные выражения

*Краткое введение*

Регулярные выражения являются мощным средством для работы с текстом. Используя регулярные выражения, можно в несколько раз сократить объем кода. Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил — сжатого описания некоторого множества строк, без необходимости перечисления всех элементов этого множества.

В JavaScript регулярные выражения являются объектами, которые принимают текс на вход методов execи test объекта RegExp или сами являются параметрами методов match, replace, search и splitобъекта String.

*Основные понятия*

**Регулярное выражение (regularexpression)** — средство для обработки строк или последовательность символов, определяющее так называемый шаблон (pattern) текста.

**Паттерн или образец (pattern)** — задает правило поиска, также иногда называют шаблоном, маской.

**Метасимволы или спецсимволы** — операции специального синтаксиса использующегося при составлении шаблонов. Это так называемые команды регулярного выражения.

**Модификаторы** — задают дополнительные условия поиска, такие как учет регистра и области поиска.

*Создание регулярного выражения*

Создать регулярное выражение можно следующими способами:

* Используя инициализатор объекта:

re = /abc\d+/

Инициализаторы объектов выполняют компиляцию регулярного выражения при вычислении скрипта. Если регулярное выражение является константным, инициализатор можно использовать для повышения производительности.

* Вызывая функцию - конструктор объекта RegExp:

re = new RegExp("abc\d+")

или

re = new RegExp(/abc\d+/)

При использовании функции-конструктора компиляция регулярного выражения происходит на этапе прогона. Если известно, что паттерн регулярного выражения будет изменяться, или если Вы заранее не знаете паттерн и он, например, задается путем пользовательского ввода. После определения регулярного выражения исходное значение изменяется, то можно использовать метод compile для компиляции нового регулярного выражения для более эффективного многократного использования.

*Задание паттерна регулярного выражения*

Паттерн регулярного выражения может состоять из простых символов, специальных символов или комбинации простых и специальных. Например, паттерн /expressionjavascript/, содержит только простые символы. Выражение вида /\d\*/, содержит только специальные символы. Шаблон/expressionjavascript\d\*/, содержит комбинацию простых и специальных символов.

*Простые символы*

При использовании простых символов для них ищется прямое соответствие. Например,/expressionjavascript/ совпадает с комбинацией символов в строке только тогда, когда подстрока 'expressionjavascript' встречается именно в таком порядке и именно с таким количеством пробелов между словами.

Совпадениебудетнайденовстроках "I'm studing expression javascript" и "I'm studing expression javascript for two days". В обоих случаях строка содержит подстроки 'expressionjavascript'. Но в строке "I'mstudingexpression **in** javascript" совпадений нет, потому что она не содержит в себе точной последовательности символов соответствующей 'expressionjavascript'.

*Специальные символы*

Использование спецсимволов дает большую гибкость, позволяя задать некое множество простых символов используемых при поиске. Например, паттерн /javascript\d\*|php\d\*/ совпадает с любой подстрокой, в которой есть слово 'javascript' или 'php', за которым следует 0 или более цифр (| означает логическое или, \d - число, а \* 0 или более раз).

Так будет найдено 'javascript' для строки "I'mstudingjavascript" и подстрока 'javascript1' для "I'mstuding javascript1". Для строк "I'mstudingphp" и "I'mstuding php50" будут найдены соответственно подстроки 'php' и 'php50'.

В таблице дан полный список - описание специальных символов, которые могут использоваться в регулярных выражениях.

**Специальные символы в регулярных выражениях**

|  |  |
| --- | --- |
| **Значение** | **Описание** |
| **\** | Изменяет интерпретацию последующего символа:   * Для символов, которые обычно рассматриваются литерально, указывает, что следующий символ является специальным и не должен интерпретироваться литерально.   Например, /d/ совпадает с символом 'd'. При помещении символа \ перед d, то есть /\d/, символ становится специальным, обозначая цифру.   * Для символов, которые обычно рассматриваются как специальные, указывает, что следующий символ не является специальным и должен интерпретироваться литерально.   Например, \* это специальный символ, который означает 0 или более совпадений с вхождением предыдущего символа; например, /a\*/ означает 0 или более символов а. Для интерпретации \* литерально, т.е. как обычного символа, необходимо поставить перед ним \; например, /a\\*/ совпадает с 'a\*'. |
| **^** | Совпадает с начало ввода или строки.  Например, /^J/ не совпадает с 'J' в строке "HernameisJane," но совпадает с первой 'J' в строке "JaneJason". |
| **$** | Совпадает с концом ввода или строки.  Например, /t$/ не совпадает с 't' в "PitJason", но совпадает в "HisnameisPit". |
| **\*** | Предшествующий символ совпадает 0 или более раз.  Например, /go\*/ совпадает с 'goo' в "A goodsample" и с 'g' в "A generalsample", но не совпадает в "A book". |
| **+** | Предшествующий символ совпадает 1 или более раз. Эквивалентно {1,}.  Например,/go+/ совпадает с 'goo' в "good", но не совпадает в "general". |
| **?** | Предшествующий символ совпадает 0 или 1 раз.  Например, /go?/ совпадает с 'go' для "goto" или "good", и с 'g' в "general". |
| **{*n*}** | Предшествующий символ совпадает точно *n* раз. Где *n* это положительное целое.  Например,/go{2}/ совпадает с 'goo' в "good" или "gooood", но не совпадает в "go". |
| **{*n*,}** | Предшествующий символ совпадает *n* и более раз. Где *n* это положительное целое.  Например,/go{2,}/ совпадает с 'goo' в "good" и с 'goooo' в "gooood", но не совпадает в "go". |
| **{*n*,*m*}** | Предшествующий символ совпадает как минимум *n*, и максимум *m* раз. Где *n* и *m* это положительные целые числа.  Например,/go{1,3}/ совпадает с 'go' в "goto", с 'goo' в "good". В "gooood" совпадает с 'gooo', т.е. с 'g' и только первыми тремя 'o', а не со всеми. |
| **(*x*)** | Совпадает с *x* и запоминает совпадение. Где *x* - выражение или набор простых символов.  Например,/(good)/ совпадает (и запоминает) 'good' в "a goodsample". Более подробно этот вопросс рассматривается в пункте ["Использование совпадений подстрок в скобках"](http://regexpres.narod.ru/documents.html#017). |
| **(?:*x*)** | Совпадает с *x*, и не запоминает совпадение. Используется для группировки символов.  Например, /(?:java|action)script/ эквивалентно /javascript|actionscript/. Совпадаетс 'javascript' встроке "studing language javascript" ис 'actionscript' встроке "studing language actionscript". |
| **(?=*x*)** | Осуществляет поиск соответствия *x* наперед, и не запоминает совпадение.  Например, /Java (?=EE|SE beta)/ совпадает с 'Java' в строке "Java EE" и в строке "Java SE beta", но нет совпадений в строке "Java ME". |
| **(?!*x*)** | Осуществляет поиск несоответствия *x* наперед, и не запоминает совпадение.  Например, /Java (?!EE|SE beta)/ совпадает с 'Java' в строке "Java ME" и не совпадает для строк "Java EE" и "Java SE beta". |
| **.** | Совпадает с любым одиночным символом, кроме символа новой строки.  Например, /.a/ совпадает с 'ca' в "car" и с 'ba' в "a baby", но не с 'a' стоящей в начале строки. Также совпадает с '2a' в "number 2a" или "v2a". |
| ***x*|*y*** | Совпадает с *x* или с *y*.  Например, /javascript|php/ совпадает с 'javascript' в "studingjavascript" и с 'php' в "studingphp" |
| **[*xyz*]** | Совпадает с любым одним из символов находящимся в скобках. Где *xyz* любые символы паттерна. Можно специфицировать диапазон символов, используя дефис.  Например, /[abcd]/ эквивалентно /[a-d]/. Совпадает с 'b' в "brisket" и с 'c' в "ache". |
| **[^*xyz*]** | Совпадение со всеми символами, кроме находящихся в скобках. Можно специфицировать диапазон символов, используя дефис.  Например, [^abc] это то же самое, что [^a-c]. Первоначально совпадает с 'r' в "brisket" и с 'h' в "chop". |
| **[\b]** | Совпадает с BackSpace. (Не путать с \b.) |
| **\b** | Совпадает с границей слова - пробел/space или символ новой строки. (Не путать с [\b].)  Например, /\bg/ совпадает с 'g' в "good", а /g\b/ совпадает с 'g' в "a goodlife", но не совпадает в "together" |
| **\B** | Совпадает с не границей слова, такой как пробел/space или символ новой строки.  Например, /\Bg/ не совпадает с 'g' в "good" и в "a goodlife", но совпадает в "together". |
| **\c*X*** | Совпадает с управляющим символом в строке. Где *X* это управляющий символ.  Например, /\cS/ совпадает с комбинацией Ctrl-S. |
| **\d** | Совпадает с цифрой. Эквивалентно [0-9].  Например, /\d\d\d/ совпадает с '101' в строке "plane F-101". |
| **\D** | Совпадает с не-цифрой. Эквивалентно [^0-9].  Например, /\D/ совпадает с 'F' в строке "F-101". |
| **\f** | Совпадает с form-feed - символом перевода формата. |
| **\n** | Совпадает с linefeed - символом перевода строки. |
| **\r** | Совпадает с carriagereturn - символом возврата каретки. |
| **\s** | Совпадает с одиночным пробельным символом, включая space, tab, formfeed, linefeed. Эквивалентно [ \f\n\r\t\v].  Например, /\sg/ совпадает с ' go' в "a goodday". |
| **\S** | Совпадает с одиночным символом, отличным от пробела. Эквивалентно [^ \f\n\r\t\v].  Например, /\Sg/ не совпадает в "a goodday", но совпадает c 'og' в "together". |
| **\t** | Совпадает с tab - символом табуляции. |
| **\v** | Совпадает с verticaltab - символом вертикальной табуляции. |
| **\w** | Совпадает с любым алфавитно-цифровым символом, включая символ подчёркивания. Эквивалентно [A-Za-z0-9\_].  Например, /\w/ совпадает с 'J' в "Java", с 'v' в "version 1.4" и с '2' в "#245." |
| **\W** | Совпадает с любым не-алфавитно-цифровым символом. Эквивалентно [^A-Za-z0-9\_].  Например, /\W/ совпадает с '%' в "100%." |
| **\*n*** | Обратная ссылка на последнее совпадение подстроки с n в скобках из регулярного выражения (с учётом левых скобок). Где *n* это положительное целое. Если количество левых скобок меньше числа *n*, то эквивалентно \0*octal*.  Например, /blue(,) red\1/ совпадаетс 'blue, red,' в "green, blue, red, silver" |
| **\0*octal*** | Где *octal* это 8-ричное значение символа.  Например, /\0144/ эквивалентно символу 'd'. |
| **\x*hex*** | Где *hex* это 16-ричное значение символа, состоящее из двух цифр.  Например, /\x70/ эквивалентно символу 'p'. |
| **\u*hex*** | Где *hex* это 16-ричное значение символа Unicode, состоящее из четырех цифр.. |

*Модификаторы*

При создании регулярного выражения можно задавать дополнительные опции поиска, также иногда называемые модификаторами. Они ставятся в конце шаблона регулярного выражения, после слеша, или как второй параметр конструктора при создании объекта RegExp и имеют следующие значения:

* i (ignorecase) - поиск без учёта регистра:

re = /abc\d+/i

или

re = new RegExp(/abc\d+/i)

или

re = new RegExp("abc\d+", "i")

Совпадает с подстрокой 'abc12' в "sample abc12" и с 'aBC12' в "sample aBC12", без учета регистра. Все три выражения эквивалентны друг другу и отличаются лишь способом их задания.

* g (globalsearch) - глобальный поиск всех совпадений:

re = new RegExp("abc\d+", "g")

Совпадает не только с 'abc12', но и с 'abc34', и с 'abc6' для строки "sample abc12 abc34 abc6 andnext". Модификатор g также может быть задан несколькими способами аналогично i.

* m (multiline) - многострочный поиск. Поддерживается не всеми браузерами.

Так же имеется возможность задания сразу нескольких опций поиска одновременно. Например, глобальный поиск без учета регистра:

re = newRegExp("abc\d+", "ig")

или

re = /abc\d+/ig

**Методы для работы с регулярными выражениями объекта String**

|  |  |
| --- | --- |
| **Название** | **Описание** |
| **Методы** | |
| **match** | Выполняет поиск совпадения. Возвращает массив информации, или null при отсутствии совпадения. |
| **search** | Тестирует на наличие совпадений в строке. Возвращает индекс совпадения или -1, если поиск завершился неудачно. |
| **replace** | Выполняет поиск совпадения в строке и заменяет найденные подстроки замещающей подстрокой. |
| **split** | Разделяет строку на массив подстрок, используя регулярное выражение или фиксированную строку в качестве значения разделителя. |

*match- тест наличия совпадений*

Назначение: выполняет поиск совпадений в строке на основании паттерна.

Синтаксис:

objString.match(aPattern);

objString - текст, в котором будет осуществляться поиск. Является экземпляром объекта String.

aPattern - паттерн (шаблон) для поиска.

Возвращаемое значение: массив совпадений или null если совпадений не найдено.

Пример:

re = new RegExp(/abc+/gi);  
vStr = "sample abcCccaBc and next";   
vArr = vStr.match(re);  
vFirstSame = vArr[0];  
vSecondSame = vArr[1];

В данном случае переменная vFirstSame равна 'abcCcc', vSecondSame равна 'aBc'.

*search- индекс первого совпадения*

Назначение: выполняет поиск первого совпадения в строке на основании паттерна.

Синтаксис:

objString.search(aPattern);

objString - текст, в котором будет осуществляться поиск. Является экземпляром объекта String.

aPattern - паттерн (шаблон) для поиска.

Возвращаемое значение: положительное целое число или -1, если совпадений не найдено.

Пример:

vStr = "sample abcccabc and next";   
vFirstIndex = vStr.search(/abc+/);

В данном случае переменная vFirstIndex равна 7.

*replace - поиск и замена совпадений*

Назначение: выполняет поиск и замену совпадений в строке на основании паттерна

Синтаксис:

objString.replace(aPattern, aSubStr);

objString - текст, в котором будет осуществляться поиск и замена. Является экземпляром объекта String.

aPattern - паттерн (шаблон) для поиска.

aSubStr - текст, которым будет заменено найденное совпадение.

Возвращаемое значение: строка с измененным текстом или с исходным текстом, если совпадений не найдено.

Пример:

vStr = "sample abccc1 abc2 and next";   
vNewStr = vStr.replace(/abc+/g,"dfg");

В данном случае переменная vNewStr будет равна "sample dfg1 dfg2 andnext".

*split - разбивка строки*

Назначение: выполняет разбиение строки на подстроки, используя в качестве разделителя паттерн или фиксированную строку.

Синтаксис:

objString.split(aPattern, aPattern|aSpliter);

objString - текст для разбиения. Является экземпляром объекта String.

aPattern - паттерн (шаблон) разделителя.

aSpliter - фиксированная строка - разделитель.

Возвращаемое значение: массив подстрок. Если совпадений не найдено, то первый элемент массива содержит исходный текст.

Пример:

vStr = "1,2,15";   
vArr = vStr.split(",");

vStr = "2\*5+9/4";   
vArr = vStr.split(/([\\*\+/-])/);

В первом примере, для разбиения используется фиксированный разделитель ",". Результирующий массивvArr будет содержать разделенные подстроки: vArr[0] равно '1', vArr[1] равно '2', vArr[2] равно '15'.

Во втором примере происходит полный разбор арифметического выражения на составляющие. Для разбиения используется регулярное выражение, которое совпадает с любым из знаков арифметических операций: умножить, сложить, поделить и отнять. Скобки в паттерне регулярного выражения нужны для запоминания самого разделителя. В итоге, результирующий массив vArr, будет содержать все выражение почленно вместе с арифметическими операциями: vArr[0] равно '2', vArr[1] равно '\*', vArr[2] равно '5' и так далее. Т.е. vArr содержит ['2', '\*', '5', '+', '9', '/', '4'].

*Использование совпадений подстрок в скобках*

Включение скобок в паттерн регулярного выражения вызывает запоминание соответствующего подсовпадения. Например, /a(b)cc/ совпадает с символами 'abcc' и запоминает 'b'. Для последующего использования этих запомненных подсовпадений используйте свойства $1, ..., $9 объекта RegExp или элементы [1], ..., [n] объекта Array, возвращаемого при поиске совпадений.

Количество подстрок в скобках не ограничено, но предопределённый объект RegExp хранит только последние 9 подстрок, а массив содержит все.

Пример использования свойств $1, ..., $9 объекта RegExp:

re = RegExp(/(c)d(f)/);   
re.exec("sample cdf");  
vFirstRem = RegExp.$1;  
vSecondRem = RegExp.$2;

В данном примере переменная vFirstRem равна 'c', а vSecondRem равна 'f'.

Пример использования массива совпадений:

re = RegExp(/(c)d(f)/);   
vArr = re.exec("samplecdf");  
vAllSame = vArr[0];  
vFirstRem = vArr[1];  
vSecondRem = vArr[2];

В данном примере переменная vFirstRem будет равна 'c', vSecondRem равна 'f', а переменная vAllSameсодержит 'cdf - полную подстроку совпадения .