Для успешной работы с регулярными выражениями, вы должны знать, что делает каждый специальный символ, флаг и метод. Это чит-лист по регулярным выражениям, к которому вы можете обращаться при попытке вспомнить, как работает метод, специальный символ или флаг.

## Определение регулярного выражения в JavaScript

Существует два способа определения регулярного выражения в JavaScript.

* var rgx = /^(\d+)/ - Вы можете использовать литерал регулярных выражений и заключить шаблон между слэшами. Это оценивается во время компиляции и обеспечивает лучшую производительность, если регулярное выражение остается постоянным.
* var rgx = new RegExp('^(\d+)') - Функция-конструктор полезна, когда регулярное выражение может программно меняться. Они компилируются во время выполнения.

## Соответствие определенному набору символов

Следующие последовательности могут использоваться для соответствия определенному набору символов.

* \w - Соответствует всем символам "слов". Символы слов являются буквенно-цифровыми (a-z, A-Z символы и подчеркивание).
* \W - Соответствует символам "не слов". Все, кроме буквенно-цифровых символов и подчеркивания.
* \d - Соответствует символам "цифр". Любая цифра от 0 до 9.
* \D - Соответствует символам "не цифр". Все, кроме с 0 до 9.
* \s - Соответствует символам пробела. Это включает пробелы, символы табуляции и разрывы строк.
* \S - Соответствует всем остальным символам, кроме пробелов.
* . - Соответствует любому символу, кроме разрывов строк.
* [A-Z] - Соответствует символам в диапазоне. Например, [A-E] будет соответствовать A, B, C, D и E.
* [ABC] - Соответствует символу в заданном наборе. Например, [AMT] будет соответствовать только A, M и T.
* [^ABC] - Соответствует всем символам, отсутствующим в заданном наборе. Например, [^A-E] будет соответствовать всем символам, кроме A, B, C, D и E.

## Указание количества символов для соответствия

Все приведенные выше выражения будут совпадать с одним символом за раз. Вы можете добавить квантификаторы, чтобы указать, сколько символов должно быть включено в совпадение сразу.

* + - Соответствует одному или нескольким вхождениям предыдущего символа. Например, \w+ вернет **ABD12D** в виде единственного соответствия вместо шести разных совпадений.
* \* - Соответствует нулю или более вхождениям предыдущего символа. Например, b\w\* соответствует полужирным частям в **b**, **bat**, **bajhdsfbfjhbe**. В целом, он соответствует нулю или более символам "слова" после "b".
* {m, n} - Соответствует не менее m и не более n вхождений предыдущего символа. {m,} будет соответствовать не менее m вхождений, и верхнего предела для совпадения не будет. {k} будет соответствовать точно k вхождениям предыдущего символа.
* ? - Соответствует нулю или одному вхождению предыдущего символа. Например, это может быть полезно при поиска двух вариантов написания для одной и той же работы. Например, /behaviou?r/ будет соответствовать как **behavior**, так и **behaviour**.
* | - Соответствует выражению до или после "pipe" символа. Например, /se(a|e)/ соответствует как see, так и sea.

## Скобочные группы в регулярных выражениях

* (ABC) - Это сгруппирует несколько символов вместе и запомнит подстроку, соответствующую им, для последующего использования. Это называется скобочной группой.
* (?:ABC) - Это также объединяет несколько символов вместе, но не запоминает совпадение. Это незапоминаемая скобочная группа.
* \d+(?=ABC) - Это будет соответствовать символу(-ам), предшествующему (?=ABC), только если за ним следует ABC. Часть ABC не будет включена в массив совпадений. Часть \d - это всего лишь пример. Это может быть любая строка регулярного выражения.
* \d+(?!ABC) - Это будет соответствовать символу(-ам), предшествующему (?!ABC), только если за ним не следует ABC. Часть ABC не будет включена в массив совпадений. Часть \d - это всего лишь пример. Это может быть любая строка регулярного выражения.

Advertisement

## Другие символы регулярных выражений

Существуют также некоторые другие символы регулярных выражений, которые не были рассмотрены в предыдущих разделах:

* ^ - Ищет регулярное выражение в начале текста или в начале каждой строки, если включен многострочный флаг.
* $ - Ищет регулярное выражение в конце текста или в конце каждой строки, если включен многострочный флаг.
* \b - Предыдущий символ соответствует, только если это граница слова.
* \B - Предыдущий символ соответствует только в том случае, если граница слова отсутствует.

## Использование флагов в регулярных выражениях

Флаги могут использоваться для контроля того, как следует интерпретировать регулярное выражение. Вы можете использовать флаги по одному или вместе в любом порядке. Это пять флагов, которые доступны в JavaScript.

* g - Искать все совпадения данного выражения в строке вместо того, чтобы возвращать только первый.
* i - Сделать поиск без учета регистра, чтобы между словами, такими как Apple, aPPLe и apple, не было разницы.
* m - Этот флаг гарантирует, что маркеры ^ и $ ищут совпадение в начале или конце каждой строки, а не всего текста.
* u - Этот флаг включает поддержку Unicode в вашем регулярном выражении.
* y - Этот флаг скажет JavaScript искать соответствия только с текущей позиции в целевой строке.

Вы можете указать флаги для регулярного выражения в JavaScript либо путем добавления их в конец литерала регулярных выражений, либо путем передачи их в конструктор RegExp. Например, /cat/i соответствует всем вхождениям **cat** независимо от случая, и RegExp ("cat", 'i') делает то же самое.

## Полезные методы регулярных выражений в JavaScript

Регулярные выражения, которые вы создаете с использованием флагов и последовательностей символов, которые мы обсуждали до сих пор, предназначены для использования с различными методами для поиска, замены или разделения строки. Вот некоторые методы, связанные с регулярными выражениями.

* test() - Проверить, содержит ли основная строка подстроку, которая соответствует шаблону, заданному данным регулярным выражением. Метод возвращает true при успешном совпадении и false в противном случае.

|  |  |
| --- | --- |
| 1 | var textA = 'I like APPles very much'; |
| 2 | var textB = 'I like APPles'; |
| 3 | var regexOne = /apples$/i |
| 5 | // Output : false |
| 6 | console.log(regexOne.test(textA)); |
| 8 | // Output : true |
| 9 | console.log(regexOne.test(textB)); |

В приведенном выше примере регулярное выражение должно искать слово **apples** только в конце строки. Вот почему мы получили false в первом случае.

* search() - Проверить, содержит ли основная строка подстроку, которая соответствует шаблону, заданному данным регулярным выражением. Он возвращает индекс совпадения при успехе и **-1** в противном случае.

|  |  |
| --- | --- |
| 1 | var textA = 'I like APPles very much'; |
| 2 | var regexOne = /apples/; |
| 3 | var regexTwo = /apples/i; |
| 5 | // Output : -1 |
| 6 | console.log(textA.search(regexOne)); |
| 8 | // Output : 7 |
| 9 | console.log(textA.search(regexTwo)); |

В этом случае первое регулярное выражение возвращает **-1**, потому что не было точного совпадения с учетом регистра.

* match() - Ищет, если основная строка содержит подстроку, которая соответствует шаблону, заданному данным регулярным выражением. Если флаг g включен, в качестве массива будут возвращены несколько совпадений.

|  |  |
| --- | --- |
| 1 | var textA = 'All I see here are apples, APPles and apPleS'; |
| 2 | var regexOne = /apples/gi; |
| 4 | // Output : [ "apples", "APPles", "apPleS" ] |
| 5 | console.log(textA.match(regexOne)); |

* exec() - Поиск, если основная строка содержит подстроку, которая соответствует шаблону, заданному данным регулярным выражением. Возвращенный массив будет содержать информацию о совпадениях и скобочных группах.

|  |  |
| --- | --- |
| 1 | var textA = 'Do you like apples?'; |
| 2 | var regexOne = /apples/; |
| 4 | // Output : apples |
| 5 | console.log(regexOne.exec(textA)[0]); |
| 7 | // Output : Do you like apples? |
| 8 | console.log(regexOne.exec(textA).input); |

* replace() - Найти подстроку, соответствующую заданному шаблону, и заменить ее на предоставленную заменяющую строку.

|  |  |
| --- | --- |
| 1 | var textA = 'Do you like aPPles?'; |
| 2 | var regexOne = /apples/i |
| 4 | // Output : Do you like mangoes? |
| 5 | console.log(textA.replace(regexOne, 'mangoes')); |

* split() - Этот метод позволит вам разбить основную строку на подстроки на основе разделителя, указанного как регулярное выражение.

|  |  |
| --- | --- |
| 1 | var textA = 'This 593 string will be brok294en at places where d1gits are.'; |
| 2 | var regexOne = /\d+/g |
| 4 | // Output : [ "This ", " string will be brok", "en at places where d", "gits are." ] |
| 5 | console.log(textA.split(regexOne)) |