Работа со строками в JavaScript

Во всех языках программирования есть такие операнды как строки. Представляют из себя простую группу символов и часто используются для передачи параметров или их хранения. К примеру значения свойств css класса могут храниться в строках или адрес сайта на который нужно будет сделать перенаправление тоже может храниться как строка.

Поэтому очень важно уметь работать со строками.

Оглавление

* [Объявление строки](http://yeap.narod.ru/js/033.html#metod-varstr)
* [Юникод](http://yeap.narod.ru/js/033.html#metod-unicode)
* [Конкатенация строк (.concat())](http://yeap.narod.ru/js/033.html#metod-concat)
* [Длина строки](http://yeap.narod.ru/js/033.html#metod-length)
* [Конвертировать число в строку (Number → String)](http://yeap.narod.ru/js/033.html#metod-convert-number-string)
* [Конвертировать строку в число (String → Number)](http://yeap.narod.ru/js/033.html#metod-convert-string-number)
* [Преобразование регистра символов](http://yeap.narod.ru/js/033.html#metod-lowercase)
* [.indexOf()](http://yeap.narod.ru/js/033.html#metod-indexOf)
* [.lastIndexOf()](http://yeap.narod.ru/js/033.html#metod-lastIndexOf)
* [.charAt()](http://yeap.narod.ru/js/033.html#metod-charAt)
* [.substring()](http://yeap.narod.ru/js/033.html#metod-substring)
* [.substr()](http://yeap.narod.ru/js/033.html#metod-substr)
* [.split()](http://yeap.narod.ru/js/033.html#metod-split)

Возьмем какую-нибудь вымышленную строку, ну, например, какую-нибудь знаменитую фразу: «JavaScript now works!». И разберем на примере этой строки. Для начала основы работы со строками.

Объявление строки

Объявление строки выглядит вот так:

var str = 'JavaScript now works!';

Сейчас была описана переменная «str» в которую мы записали определенную строку, заключив ее между одинарных кавычек. Если бы в строке уже присутствовали одинарные кавычки, которые являлись бы частью строки, то необходимо было бы использовать экранирование кавычек. То есть, ставить перед внутренней кавычкой обратный слеш «**\**».

var str = 'JavaScript \'now\' work!';

Кстати, вложенные двойные кавычки никак не будут конфликтовать и их не нужно экранировать.

var str = 'JavaScript "now" works!';

Обратная вложенность, одинарные в двойных, тоже совершенно нормально вместе сосуществуют.

Юникод

Объявим еще две строки «str1» и «str2» в которые запишем по два разных символа. Они тоже будут строками, потому что в джаваскрипте любые текстовые данные имеют тип строка (String), и нету никаких отдельных литер, как в других языках.

var str1 = 'а';

var str2 = 'б';

alert (str1 < str2); // вернет true

Вроде бы это логично, т.к. в кириллическом алфавите буква «а» стоит раньше буквы «б». Но вот если мы возьмем буквы «ё» и «я», то выражение:

alert ('ё' < 'я'); // вернет false

будет неправильным.

Это происходит из-за того, что джаваскрипту плевать какие алфавиты используют люди. У него есть свой «юникодный алфавит» в котором каждый символ соответствует определенному коду. И в выражениях выше, он сравнивает не буквы, а их представления в юникоде. При этом не имеет совершенно никакого значения в какой кодировке находится документ с которым вы работаете в windows-1251 или в utf-8 или может еще в какой-нибудь экзотической, не важно, все свои символы джаваскрипт переведет в свой юникод.

Получить юникод символа можно с помощью метода .charCodeAt(pos). Где «pos» это номер элемента в строке, начиная с нуля.

var str = 'JavaScript now works!';

var str.charCodeAt(4) // вернет «83»

var str.charCodeAt(0) // вернет «74»

var str.charCodeAt(11) // вернет «110»

Коды символов учитывают и регистр, то есть «а» - малая и «А» - большая будут иметь разные коды. В примере выше, выведенные юникоды будут такими: «83», «74», «111» и соответственно в строке «str» они будут равны 5-му, 1-му, и 12-му символам, то есть «S», «J» и «o».

Это не сложно проверить с помощью метода .fromCharCode, вот так:

String.fromCharCode(83,74,111) // вернет «SJo»

Если посчитать символы в строке, то как раз большая «S» будет пятым символом строки, а «J» большая первым и т.д.

Конкатенация строк

С помощью оператора «+» строки можно складывать. Например, у нас есть две строки, «js» и «phrase» со следующими значениями:

var js = 'JavaScript';

var phrase = 'now works!';

var str = js+' '+phrase;

В результате в «str» мы получим нашу исходную строку, надо еще заметить, что в сложении, если присмотреться участвовал пробел, иначе бы у нас слова «JavaScript» и «now» склеились.

Для такой же операции существует также и метод – .concat().

var js = 'JavaScript';

var phrase = 'now works!';

var str = js.concat(' ').concat(phrase);

Здесь также в конкатенации участвует пробел.

Длина строки

Так как, в джаваскрипте всё можно рассматривать в качестве объекта, то и строка не исключение. Поэтому с помощью встроенной функции .length можно узнать длину строки.

var str = 'JavaScript now works!';

str.length; //вернет 21, т.к. в строке 21 символов

Так можно вычислить длину любой строки, например, адрес этой страницы в интернете, можно представить, как строку и вывести её длину:

var url = document.location.href; //это «http://yeap.narod.ru/js/033.html»

str.length;//вернет 32

т.к. в строке «http://yeap.narod.ru/js/033.html» 32 символа

Конвертировать число в строку (Number → String)

Как было сказано выше, любой текст уже является строкой. В строки можно преобразовывать и числа, с помощью метода .toString(). Например, у нас есть какое-нибудь число — numb.

var numb = 12; // это число

numb.toString(); //а теперь это строка

Проверить, что произошло преобразование довольно легко. Допустим, у нас есть все тоже числоnumb и мы захотим произвести с ним какую-нибудь математическую операцию, ну, например сложение:

var numb = 12;

var result = numb + 3;

alert(result) // вернет 15

var result = numb.toString() + 3;

alert (result) // вернет 123

В примере выше, во второй строчке к числу numb с помощью оператора «**+**» мы добавили «3» и, совершенно, ожидаемо получили на следующей строке число «15». А вот далее к numb был применен метод .toString() и nubm был преобразован в строку. Далее джаваскрипт увидел, что к строке пытаются добавить число, а как было сказано выше в случае строками оператор «**+**» при работе со строками выполняет конкатенацию, то есть просто склеивает строки, в нашем случае мы логично получили строку «123».

Конвертировать строку в число (String → Number)

Некоторые строки можно наоборот преобразовывать в числа. Так как, в примере выше, видно, что оператор сложения плюс (**+**) при работе со строками склеивает их. А вот оператор умножения (**\***), остается верен математике и честно пытается умножать.

Все знают, что если умножить любое число на единицу, в результате получится тоже самое число.

var str = '12';// это строка

var result = str \* 1; // умножаем на 1

alert(result) // а теперь это число 12

Тут джаваскрипт увидел, что строку '12' можно без проблем преобразовать в число, а затем умножить на единицу. И таким образом исходная строка превратится в число. А если в строке есть литеры в итоге JavaScript получит не число тип NaN (не число — Not a Number).

В этом случае можно воспользоваться встроенным методом .parseInt():

var str = '123абв';

var result = str.parseInt(); // вернет «123»

В примере выше видно, что JavaScipt отбросил все лишнее от строки взял только цифры и преобразовал их в число (Number).

Методы для работы со строками

На примере длины строки или других методов, которые были описаны выше, можно заметить, что методы со строкой работают по такому шаблону:

имя\_строки.имя\_метода()

Преобразование регистра символов

С помощью методов .toUpperCase() и .toLowerCase() можно менять регистр символов строк.

var str = 'JavaScript now works!';

var result = str.toUpperCase(); // вернет «JAVASCRIPT NOW WORKS!»

var str = 'JavaScript now works!';

var result = str.toLowerCase(); // вернет «javascript now works!»

.indexOf()

Один из наиболее часто используемых методов для работы со строками – .indexOf(). Каждый символ в строке имеет свою позицию (свой индекс). И с помощью этого метода можно получить индекс символа:

var str = 'JavaScript now works!';

var result = str.indexOf('J'); // вернет «0»

var result = str.indexOf('S'); // вернет «4»

Тут как видно из первого примера отсчет индексов также начинается с нуля.

Если в строке содержится несколько одинаковых символов, то метод вернет позицию первого встретившегося с начала строки. Также можно задать сразу несколько символов: .indexOf('J S'). В этом случае метод вернет 0, т.к. первый символ «J» имеет индекс 0.

Метод можно использовать еще и таким образом:

var result = str.indexOf('a', 3); // вернет «3»

Вторым параметром здесь указано с какого индекса начать поиск символа 'a' вторая буква 'a' в строке str имеет как раз индекс 3.

Если символ указанный символ не встречается в строке то вернется значение -1 (значит false)

.lastIndexOf()

Он работает аналогично .indexOf() только выполняет поиск символа, начиная с конца строки.

var str = 'JavaScript now works!';

var result = str.lastIndexOf('o', 3); // вернет «16»

Последняя встречающаяся буква 'o' в строке str имеет индекс 16.

.charAt()

Метод .charAt() вернет символ, который будет соотвествовать указанной позиции.

var str = 'JavaScript now works!';

var result = str.charAt(0); // вернет «J»

var result = str.charAt(16); // вернет «o»

var result = str.charAt(2); // вернет «v»

Надо опять же помнить, что первый символ имеет индекс «0».

.substring()

Метод .substring() для получения подстроки. Он принимает два параметра в виде двух индексов – первого и последнего и возвращает все символы стоящие между этими индекса включая указанные.

var str = 'JavaScript now works!';

var result = str.substring(0, 3); // вернет «Java»

Также в методе можно указать один параметр и тогда он вернет строку начиная с указанного символа и до конца.

var result = str.substring(4); // вернет «Script now works!»

.substr()

Метод .substr() работает несколько иначе. В качестве второго параметра он принимает не индекс позиции символа, а количество символов.

var result = str.substring(4, 6); // вернет «Script»

var result = str.substring(0, 4); // вернет «Java»

.split()

Метод .split() служит для разбития строки на части через указанный символ. Возьмем все ту же строку str и попробуем ее разделить на три части через пробел между словами. Пробел ведь тоже символ.

var str = 'JavaScript now works!';

var result = str.split(' ');

Переменная result теперь станет масивом с тремя элементами: «JavaScript», «now» и «works!».

Пробел исчез, а выводить часть строки можно теперь так:

result[0]; // вернет «JavaScript»

result[1]; // вернет «now»

result[2]; // вернет «works!»

JavaScript: методы работы со строками

Когда пишешь **JavaScript**, очень часто приходится лазить по интернету в поисках информации о синтаксисе и параметрах для методов, работающих со строками.

Я прочитал множество статей по работе со строками. В данном посте будет показаны примеры и краткие описания наиболее распространенных методов по работе со строками. Я попытался поставить самые частые методы в верхней части для быстрого ознакомления.

Конечно, большинство опытных разработчиков уже достаточно знакомы со многими из методов, но я думаю, что это хороший список для начинающих, чтобы понимать спектр методов, которые могут помочь выполнить сложные операции простыми средствами.

Конвертирование в String

Вы можете конвертировать число, булево выражение или объект в строку:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var myNumber = 24; // 24  var myString = myNumber.toString(); // "24" |

Вы можете сделать это так же с помощью **String()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var myNumber = 24; // 24  var myString = String(myNumber); // "24" |

Если вы не уверены, что значение не является **null** или **undefined**, вы можете использовать **String()**, которая всегда возвращает строку, независимо от типа значения.

Разделение строки в подстроки

Чтобы разделить строки в массив подстрок, вы можете использовать метод **split():**

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var myString = "coming,apart,at,the,commas";  var substringArray = myString.split(","); // ["coming", "apart", "at", "the", "commas"]  var arrayLimited = myString.split(",", 3); // ["coming", "apart", "at"] |

Как видно в последней строке, второй параметр функции — это лимит количества элементов, которое будет в итоговом массиве.

Получение длины строки

Чтобы найти, сколько символов в строки, мы используем свойство **length:**

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var myString = "You're quite a character.";  var stringLength = myString.length; // 25 |

Поиск подстроки в строке

Есть два метода для поиска подстроки:

Использование **indexOf()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "Johnny Waldo Harrison Waldo";  var wheresWaldo = stringOne.indexOf("Waldo"); // 7 |

**indexOf()** метод начинает поиск подстроки с начала строки, и возвращает позицию начала первого вхождения подстроки. В данном случае — 7 позиция.

Использование**lastIndexOf()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "Johnny Waldo Harrison Waldo";  var wheresWaldo = stringOne.lastIndexOf("Waldo"); // 22 |

Метод возвращает начальную позицию последнего вхождения подстроки в строку.

В обоих методах, если подстрока не найдена, возвращается значение -1, и оба принимают необязательный второй аргумент, указывающий положение в строке, где вы хотите начать поиск. Таким образом, если второй аргумент «5», **indexOf()** начинает поиск с 5 символа, игнорируя символы 0-4, в то время как **lastIndexOf()** начинает поиск с символа 5 и идет в обратном направлении, игнорируя символы 6 и дальше.

Замена подстроки

Чтобы заменить вхождение подстроки в строке на другую подстроку, вы можете использовать **replace()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var slugger = "Josh Hamilton";  var betterSlugger = slugger.replace("h Hamilton", "e Bautista");  console.log(betterSlugger); // "Jose Bautista" |

Первый аргумент — то, что вы хотите заменить и второй аргумент — новая строка. Функция заменяет только первое вхождение подстроки в строку.

Чтобы заменить все вхождения, нужно использовать регулярное выражение с глобальным флагом:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var myString = "She sells automotive shells on the automotive shore";  var newString = myString.replace(/automotive/g, "sea");  console.log(newString); // "She sells sea shells on the sea shore" |

Второй аргумент может включать специальный шаблон или функцию. Подробней можно почитать [здесь](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String/replace).

Получить символ по заданной позиции в строке

Получить символ мы можем с помощью функции **charAt()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var myString = "Birds of a Feather";  var whatsAtSeven = myString.charAt(7); // "f" |

Как часто бывает в JavaScript, первая позиция в строке начинается с 0, а не с 1.

В качестве альтернативной функции можно использовать **charCodeAt()** функцию, которая код символа.

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var myString = "Birds of a Feather";  var whatsAtSeven = myString.charCodeAt(7); // "102"  var whatsAtEleven = myString.charCodeAt(11); // "70" |

Заметьте, что код для символа «F» (11 позиция) другой, нежели у символа «f» (позиция 7).

Соединение строк

В большинстве случаем, чтобы соединить строки, можно использовать оператор «+». Но так же можно использовать метод **concat()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "Knibb High football ";  var stringTwo = stringOne.concat("rules."); // "Knibb High football rules" |

Таким способом мы можем соединить множество строк в одну в том порядке, в котором они записаны:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var stringOne = "Knibb ";  var stringTwo = "High ";  var stringThree = "football ";  var stringFour = "rules.";  var finalString = stringOne.concat(stringTwo, stringThree, stringFour);  console.log(finalString); // "Knibb high football rules." |

Извлечение подстроки

Есть 3 способа получения строки из части другой строки:

Используя **slice()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "abcdefghijklmnopqrstuvwxyz";  var stringTwo = stringOne.slice(5, 10); // "fghij" |

Используя **substring()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "abcdefghijklmnopqrstuvwxyz";  var stringTwo = stringOne.substring(5, 10); // "fghij" |

В обеих функция первый параметр — символ, с которого начинает подстрока (начиная с 0 позиции) и второй аргумент (необязательный) — позиция символа, до которого возвращается подстрока. В примере (5, 10) возвращается строка между позицией 5 и 9.

Используя **substr()**:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var stringOne = "abcdefghijklmnopqrstuvwxyz";  var stringTwo = stringOne.substr(5, 10); // "fghijklmno" |

Первый аргумент — позиция символа, с которого начинается новая строка и второй аргумент — количество символов от начальной позиции новой строки. Т.е. (5, 10) возвращает 10 символов, начиная с 5 позиции.

Перевод строки в верхний или нижний регистр.

Есть 4 метода для перевода. Первые 2 переводят строку в верхний регистр:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var stringOne = "Speak up, I can't hear you.";  var stringTwo = stringOne.toLocaleUpperCase(); // "SPEAK UP, I CAN'T HEAR YOU"  var stringThree = stringOne.toUpperCase(); // "SPEAK UP, I CAN'T HEAR YOU" |

Другие 2 переводят строку в нижний регистр:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3 | var stringOne = "YOU DON'T HAVE TO YELL";  var stringTwo = stringOne.toLocaleLowerCase(); // "you don't have to yell"  var stringThree = stringOne.toLowerCase(); // "you don't have to yell" |

Лучше использовать «locale» методы, т.к. в разных местах, например, в Турции отображение регистров работает не совсем так, как мы привыкли и поэтому результат может быть тот, который мы хотели. Если использовать «locale» методы, то таких проблем не будет.

[Ссылка MDN](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String/toLocaleLowerCase)

Pattern Matching

Соответствие по шаблону в строке может быть использовано с помощью 2-х методов, которые работают по-разному.

Метод **match()** применяется к строке и он принимает в качестве параметра регулярное выражение:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | var myString = "How much wood could a wood chuck chuck";  var myPattern = /.ood/;  var myResult = myString.match(myPattern); // ["wood"]  var patternLocation = myResult.index; // 9  var originalString = myResult.input // "How much wood could a wood chuck chuck" |

Метод **exec()** применяется к объекту регулярного выражения и принимает в качестве параметра строку:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | var myString = "How much wood could a wood chuck chuck";  var myPattern = /.huck/;  var myResult = myPattern.exec(myString); // ["chuck"]  var patternLocation = myResult.index; // 27  var originalString = myResult.input // "How much wood could a wood chuck chuck" |

В обоих методах возвращается лишь первое совпадение. Если совпадений не было   — возвращается **null.**

Так же можно использовать метод **search()**, который принимает регулярное выражение и возвращает позицию первого совпадения по шаблону:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2 | var myString = "Assume";  var patternLocation = myString.search(/ume/); // 3 |

Если совпадений не было — возращается «**-1**«.

Сравнение двух строк для сортировки

Вы можете сравнить 2 строки, чтобы определить, какая их них идет первая по алфавиту. Для этого воспользуемся методом **localeCompare()**, который возвращает 3 возможных значения:

[?](http://alexdev.ru/398/)

|  |  |
| --- | --- |
| 1  2  3  4  5 | var myString = "chicken";  var myStringTwo = "egg";  var whichCameFirst = myString.localeCompare(myStringTwo); // -1 (Chrome returns -2)  whichCameFirst = myString.localeCompare("chicken"); // 0  whichCameFirst = myString.localeCompare("apple"); // 1 (Chrome returns 2) |

Как было показано выше, отрицательное число возвращается, если строковый аргумент идет после исходной строки. Положительное число, если строковый аргумент идет раньше исходной строки. Если вернулся **0** — значит строки равны.

Для проверки возвращаемого значения лучше использовать if ( result < 0 ), чем if ( result === -1 ). Последнее не будет работать в Chrome.

И напоследок несколько интересных ссылок:  [Строки MDN](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String), [строковые объекты](http://msdn.microsoft.com/en-us/library/ecczf11c(v=vs.94)).

Спасибо за внимание, надеюсь, что вы узнали много нового и интересного!