

Строки

String in JavaScript 'Это то же строка'

- Строка это последовательность символов
 - Текст оборачивается в одинарные (' ') или двойные кавычки ("")

```
var str1 = "Текст, сохранный в переменной";
var str2 = 'Текст заключенный в одинарные кавычки';
```

- Строка это **примитивный тип**
 - он копирует / хранит по значению
- Строка так же **неизменяемы тип**
 - Всякий раз, когда строка меняется, копируется новая строка

Обертка строки new String

- Строка это примитивный тип, она имеет объект обертку
- Примитивный тип хранит только значение
 - Когда свойство вызывается, движок JS конвертирует примитив в соответствующий ему тип объекта и вызывает свойство
- Since primitive type wrappers are of type object, properties can be attached to them

```
let str = 'пример';
str.length;
```

```
let str = 'пример';
let tempStr = new String(str);
tempStr.length;
```

От объекта к примитивному типу

- JavaScript имеет простой синтаксис
 - От string до number
- Переход от объекта к примитиву включает
 - new String('...') - создается строчный объект
 - String(strObject) - создается строчный примитив

```
let base = 'string';
let strObj = new String(base);
let str = String(strObj);
```

Методы строк

- string.length
 - Возвращает число символов в строке
- Индексация (string[index])
 - Получает один символ из строки по его индексу
 - Если индекс выходит за пределы диапазона индекса символов, возвращается undefined
 - string[-1] or string[string.length]
- charAt(index)
 - получает один символ из строки на месте индекса
 - Сильно похож на Индексацию
- string.concat(string2)
 - Возвращает новую строку, состоящую из двух строк
- string.replace(str1, str2)
 - Заменяет вхождение str1 в str2
- string.search(regex)
 - Поиск подстроки с использованием регулярного выражения
- string.indexOf(substring [,position])
 - Возвращает **крайний левый** вхождение substring в строку следующую после position
 - По умолчанию позиция начинается с 0 индекса
 - Если строка не содержит substring, возвращается -1
- string.lastIndexOf(substring [,position])
 - Возвращает **крайнее правое** вхождение substring в строку которое предшествует position
 - Позиция опциональная, значение по умолчанию string.length
 - Если строка не содержит подстроку substring, возвращает -1
- string.split(separator)
 - Разделяет строку по separator и возвращает массив строк, содержащий разделенные части
 - Сепаратор может быть регулярным выражением
- string.trim()
 - Удаляет пробелы в начале и конце строки
- str.trimLeft(), str.trimRight()
 - Удаляет пробелы с лева / с права от строки
- string.substr(start, length)
 - Возвращает подстроку начинающуюся со start и заканчивающуюся через length символов
 - length опционально
- string.substring(start, end)
 - Возвращает подстроку начинающуюся с позиции start и заканчивающуюся позицией end
- string.valueOf()
 - Возвращает примитивное значение строки

Конкатенация строк 'Имя' + ' ' + 'Фамилия'

- Строка это неизменяемый тип
 - Значение не может быть изменено
 - Вместо этого создается новая строка
- Существует несколько способов конкатенации строк

```
var strConcat1 = str1 + str2;
var strConcat2 = str.concat(str2);
```

- Объединение строк это ресурсозатратная
 - Каждая конкатенация выделяет память
- Объединение строк это наиболее часто используемая операция
 - Тем не менее достаточно сложно оптимизировать эту задачу
 - Каждый браузер оптимизирует эту операцию по своему усмотрению
 - Старые браузер очень медленно конкатенируют используя +
- Если вы поддерживаете старые браузеры то используйте `array.join("")` для объединения строк
- Работает так же как построитель строки
 - Медленнее в современных браузерах

```
[].push(str1, str2, str3, ...).join('');
```

Управляющие символы

- Как на счет управляющих последовательностей?
 - Замена зарезервированных символов их управляющими последовательностями
 - Превентивная JavaScript внедрение
- При использовании JavaScript на стороне клиента, зарезервированными символами являются `<`, `>`, `&`, `'` и `"`

```
var script = document.createElement('script');
script.innerHTML =
  'document.location = \'http://bad_place.com\'';
document.body.appendChild(script);
```

- Выполняется когда происходит замена зарезервированного символа, его управляющей последовательностью
 - Может быть добавлено к прототипу строки

```
String.prototype.htmlEscape = function () {
  let escapedStr = String(this)
    .replace(/&/g, '&amp;');
    .replace(/</g, '&lt;');
    .replace(/>/g, '&gt;');
    .replace(/"/g, '&quot;');
    .replace(/'/g, '&#39;');
  return escapedStr;
}
```

Расширения строковых методов

- Обрезание строки
 - `string.trim()`, `string.trimLeft()`, `string.trimRight()`
 - Поддерживается всеми современными браузерами
 - Для старых браузеров используйте shim
 - `string.trimChars(chars)`, `string.trimLeftChars(chars)`, `string.trimRightChars(chars)`
 - Удалить не-пробельные символы
 - Нет нативной реализации
- Отступ строки
- `str.padLeft(count [,char])`, `str.padRight(count [,char])`
 - Сдвигает строку влево или вправо
 - Заполняет пространство пробелами или другими символами
 - Нет собственной реализации

```
String.prototype.padLeft = function (count, char) {
  char = char || ' ';
  if(char.length > 1) return String(this);
  if(str.length >= count) return String(this);
  var s = String(this);
  for (var i = 0, len = this.len; i < count - len; i++) {
    s = char + s;
  }
  return s;
}
```