DOM. Документ и объект страницы

Оглавление

[Операторы и операнды 0](#_Toc447573385)

[Приоритет операций 1](#_Toc447573386)

[Побитовые операторы 6](#_Toc447573387)

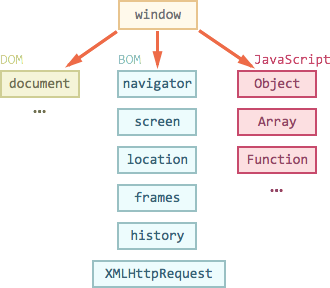
[Используемая литература 7](#_Toc447573388)

# Дерево DOM

Сам по себе язык JavaScript не предусматривает работы с браузером.

Он вообще не знает про HTML. Но позволяет легко расширять себя новыми функциями и объектами.

На рисунке ниже схематически отображена структура, которая получается если посмотреть на совокупность браузерных объектов с «высоты птичьего полёта».



Как видно из рисунка, на вершине стоит window.

У этого объекта двоякая позиция – он с одной стороны является глобальным объектом в JavaScript, с другой – содержит свойства и методы для управления окном браузера, открытия новых окон, например:

|  |
| --- |
| // открыть новое окно/вкладку с URL http://ya.ru  window.open('http://ya.ru'); |

Глобальный объект document даёт возможность взаимодействовать с содержимым страницы.

Пример использования:

|  |
| --- |
| document.body.style.background = 'red';  alert( 'Элемент BODY стал красным, а сейчас обратно вернётся' );  document.body.style.background = ''; |

Он и громадное количество его свойств и методов описаны в [стандарте W3C DOM](http://www.w3.org/DOM/DOMTR).

По историческим причинам когда-то появилась первая версия стандарта DOM Level 1, затем придумали ещё свойства и методы, и появился DOM Level 2, на текущий момент поверх них добавили ещё DOM Level 3 и готовится DOM 4.

Современные браузеры также поддерживают некоторые возможности, которые не вошли в стандарты, но де-факто существуют давным-давно и отказываться от них никто не хочет. Их условно называют «DOM Level 0».

Также информацию по работе с элементами страницы можно найти в стандарте [HTML 5](http://www.w3.org/TR/html5/Overview.html).

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов.

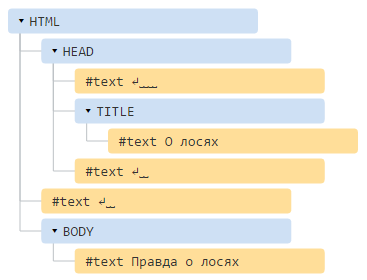
Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

Построим, для начала, дерево DOM для следующего документа.

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <title>О лосях</title>  </head>  <body>  Правда о лосях  </body>  </html> |

Его вид:



В этом дереве выделено два типа узлов.

1. Теги образуют *узлы-элементы* (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.
2. Текст внутри элементов образует *текстовые узлы* (text node), обозначенные как #text. Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

Обратите внимание на специальные символы в текстовых узлах:

* перевод строки: ↵
* пробел: ␣

**Пробелы и переводы строки – это тоже текст, полноправные символы, которые учитываются в DOM.**

В частности, в примере выше тег <html> содержит не только узлы-элементы <head> и <body>, но и#text (пробелы, переводы строки) между ними.

Впрочем, как раз на самом верхнем уровне из этого правила есть исключения: пробелы до <head>по стандарту игнорируются, а любое содержимое после </body> не создаёт узла, браузер переносит его внутрь, в конец body.

Зачем, кроме красивых рисунков, нужна иерархическая модель DOM?

**DOM нужен для того, чтобы манипулировать страницей – читать информацию из HTML, создавать и изменять элементы.**

Узел HTML можно получить как document.documentElement, а BODY – как document.body.

Получив узел, мы можем что-то сделать с ним.

Например, можно поменять цвет BODY и вернуть обратно:

|  |
| --- |
| document.body.style.backgroundColor = 'red';  alert( 'Поменяли цвет BODY' );  document.body.style.backgroundColor = '';  alert( 'Сбросили цвет BODY' ); |

DOM предоставляет возможность делать со страницей всё, что угодно.

Позже мы более подробно рассмотрим различные свойства и методы DOM-узлов.

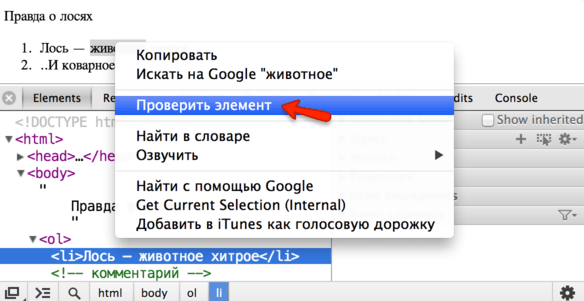
# Работа с DOM из консоли

Исследовать и изменять DOM можно с помощью инструментов разработки, встроенных в браузер. Посмотрим средства для этого на примере Google Chrome.

Откройте любой документ и, в инструментах разработчика, перейдите во вкладку Elements.

Чтобы проанализировать любой элемент:

* Выберите его во вкладке Elements.
* …Либо внизу вкладки Elements есть лупа, при нажатии на которую можно выбрать элемент кликом.
* …Либо, что обычно удобнее всего, просто кликните на нужном месте на странице правой кнопкой и выберите в меню «Проверить Элемент».



Справа будет различная информация об элементе:

**Computed Style**

Итоговые свойства CSS элемента, которые он приобрёл в результате применения всего каскада стилей, включая внешние CSS-файлы и атрибут style.

**Style**

Каскад стилей, применённый к элементу. Каждое стилевое правило отдельно, здесь же можно менять стили кликом.

**Metrics**

Размеры элемента.

…

И еще некоторые реже используемые вкладки, которые станут понятны по мере изучения DOM.

Зачастую бывает нужно выбрать элемент DOM и сделать с ним что-то на JavaScript.

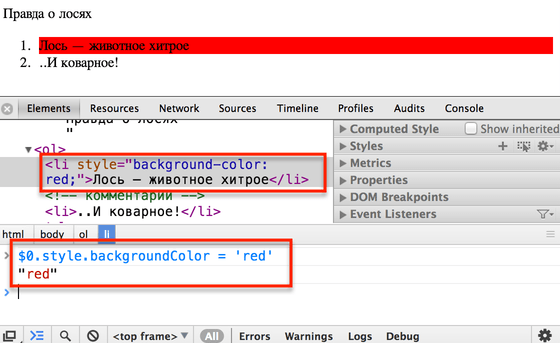
Находясь во вкладке Elements, откройте консоль нажатием Esc (или перейдите на вкладку Console).

**Последний элемент, выбранный во вкладке Elements, доступен в консоли как $0, предыдущий – $1 и так далее.**

Запустите на элементе команду, которая делает его красным:

|  |
| --- |
| $0.style.backgroundColor = 'red'; |

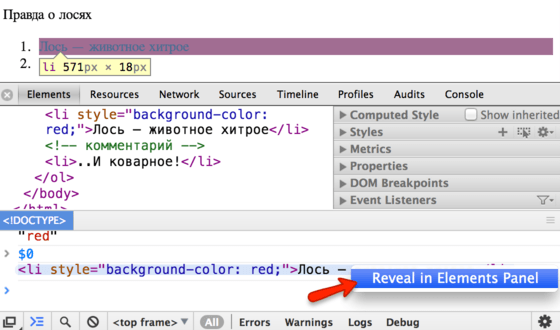
В браузере это может выглядеть примерно так:



Мы выделили элемент, применили к нему JavaScript в консоли, тут же увидели изменения в браузере.

Есть и обратная дорожка. Любой элемент из JS-переменной можно посмотреть во вкладке Elements, для этого:

1. Выведите эту переменную в консоли, например при помощи console.log.
2. Кликните на выводе в консоли правой кнопкой мыши.
3. Выберите соответствующий пункт меню.



Таким образом, можно легко перемещаться из Elements в консоль и обратно.

Для поиска элементов в консоли есть два специальных метода:

* $$("div.my") – ищет все элементы в DOM по данному CSS-селектору.
* $("div.my") – ищет первый элемент в DOM по данному CSS-селектору.

Более полная документация по методам консоли доступна на страницах [Console API Reference для Chrome](https://developers.google.com/web/tools/chrome-devtools/debug/console/console-reference) и [Command Line API для Firebug](https://getfirebug.com/wiki/index.php/Command_Line_API), а также на [firebug.ru](http://firebug.ru/).

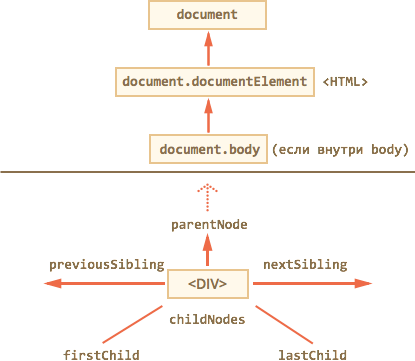
Другие браузеры реализуют похожий функционал, освоив Chrome/Firebug, вы легко с ними разберётесь.

# Навигация по DOM

DOM позволяет делать что угодно с HTML-элементом и его содержимым, но для этого нужно сначала нужный элемент получить.

Доступ к DOM начинается с объекта document. Из него можно добраться до любых узлов.

Так выглядят основные ссылки, по которым можно переходить между узлами DOM:



Посмотрим на них повнимательнее.

**Сверху documentElement и body**

Самые верхние элементы дерева доступны напрямую из document.

**<HTML> = document.documentElement**

Первая точка входа – document.documentElement. Это свойство ссылается на DOM-объект для тега<html>.

**<BODY> = document.body**

Вторая точка входа – document.body, который соответствует тегу <body>.

В современных браузерах (кроме IE8-) также есть document.head – прямая ссылка на <head>

Здесь и далее мы будем использовать два принципиально разных термина.

* **Дочерние элементы (или дети)** – элементы, которые лежат *непосредственно* внутри данного. Например, внутри <HTML> обычно лежат <HEAD> и <BODY>.
* **Потомки** – все элементы, которые лежат внутри данного, вместе с их детьми, детьми их детей и так далее. То есть, всё поддерево DOM.

Псевдо-массив childNodes хранит все дочерние элементы, включая текстовые.

Пример ниже последовательно выведет дочерние элементы document.body:

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <body>  <div>Начало</div>  <ul>  <li>Информация</li>  </ul>  <div>Конец</div>  <script>  for (var i = 0; i < document.body.childNodes.length; i++) {  alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT  }  </script>  ...  </body>  </html> |

Обратим внимание на маленькую деталь. Если запустить пример выше, то последним будет выведен элемент <script>. На самом-то деле в документе есть ещё текст (обозначенный троеточием), но на момент выполнения скрипта браузер ещё до него не дошёл.

Пробельный узел будет в *итоговом документе*, но его еще нет на момент выполнения скрипта.

Свойства firstChild и lastChild обеспечивают быстрый доступ к первому и последнему элементу.

При наличии дочерних узлов всегда верно:

|  |
| --- |
| elem.childNodes[0] === elem.firstChild  elem.childNodes[elem.childNodes.length - 1] === elem.lastChild |

DOM-коллекции, такие как childNodes и другие, которые мы увидим далее, не являются JavaScript-массивами.

В них нет методов массивов, таких как forEach, map, push, pop и других.

|  |
| --- |
| var elems = document.documentElement.childNodes;  elems.forEach(function(elem) { // нет такого метода!  /\* ... \*/  }); |

Именно поэтому childNodes и называют «коллекция» или «псевдомассив».

Можно для перебора коллекции использовать обычный цикл for(var i=0; i<elems.length; i++) ... Но что делать, если уж очень хочется воспользоваться методами массива?

Это возможно, основных варианта два:

Применить метод массива через call/apply:

|  |
| --- |
| var elems = document.documentElement.childNodes;  [].forEach.call(elems, function(elem) {  alert( elem ); // HEAD, текст, BODY  }); |

При помощи Array.prototype.slice сделать из коллекции массив.

Обычно вызов arr.slice(a, b) делает новый массив и копирует туда элементы arr с индексами от a до b-1 включительно. Если же вызвать его без аргументов arr.slice(), то он делает новый массив и копирует туда все элементы arr.

Это работает и для коллекции:

|  |
| --- |
| var elems = document.documentElement.childNodes;  elems = Array.prototype.slice.call(elems); // теперь elems - массив  elems.forEach(function(elem) {  alert( elem.tagName ); // HEAD, текст, BODY  }); |

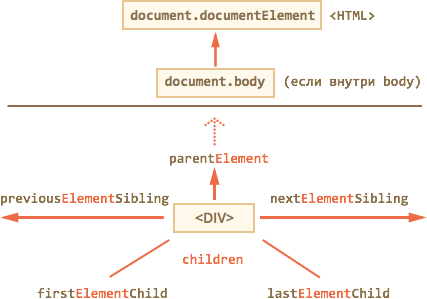
Доступ к элементам слева и справа данного можно получить по ссылкам previousSibling /nextSibling.

Родитель доступен через parentNode. Если долго идти от одного элемента к другому, то рано или поздно дойдёшь до корня DOM, то есть до document.documentElement, а затем и document.

Навигационные ссылки, описанные выше, равно касаются всех узлов в документе. В частности, вchildNodes сосуществуют и текстовые узлы и узлы-элементы и узлы-комментарии, если есть.

Но для большинства задач текстовые узлы нам не интересны.

Поэтому посмотрим на дополнительный набор ссылок, которые их не учитывают:



Эти ссылки похожи на те, что раньше, только в ряде мест стоит слово Element:

* children – только дочерние узлы-элементы, то есть соответствующие тегам.
* firstElementChild, lastElementChild – соответственно, первый и последний дети-элементы.
* previousElementSibling, nextElementSibling – соседи-элементы.
* parentElement – родитель-элемент.

Модифицируем предыдущий пример, применив children вместо childNodes.

Теперь он будет выводить не все узлы, а только узлы-элементы:

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <body>  <div>Начало</div>  <ul>  <li>Информация</li>  </ul>  <div>Конец</div>  <script>  for (var i = 0; i < document.body.children.length; i++) {  alert( document.body.children[i] ); // DIV, UL, DIV, SCRIPT  }  </script>  ...  </body>  </html> |

# Свойства узлов

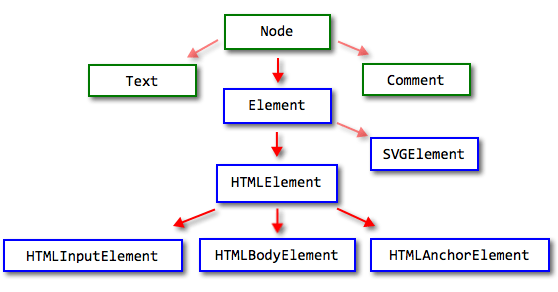
Самое главное различие между DOM-узлами – разные узлы являются объектами различных классов.

Поэтому, к примеру, у узла, соответствующего тегу <td> – одни свойства, у <form> – другие, у <a> – третьи.

Есть и кое-что общее, за счёт наследования.

Классы DOM образуют иерархию.

Основной объект в ней: [Node](http://dom.spec.whatwg.org/#interface-node), от которого наследуют остальные:



На рисунке выше изображены основные классы:

* Прямо от Node наследуют текстовые узлы Text, комментарии Comment и элементы Element.
* Элементы Element – это ещё не HTML-элементы, а более общий тип, который используется в том числе в XML. От него наследует SVGElement для SVG-графики и, конечно, HTMLElement.
* От HTMLElement уже наследуют разнообразные узлы HTML:
  + Для <input> – HTMLInputElement
  + Для <body> – HTMLBodyElement
  + Для <a> – HTMLAnchorElement… и так далее.

Узнать класс узла очень просто – достаточно привести его к строке, к примеру, вывести:

|  |
| --- |
| alert( document.body ); // [object HTMLBodyElement] |

Можно и проверить при помощи instanceof:

|  |
| --- |
| alert( document.body instanceof HTMLBodyElement ); // true  alert( document.body instanceof HTMLElement ); // true  alert( document.body instanceof Element ); // true  alert( document.body instanceof Node ); // true |

Как видно, DOM-узлы – обычные JavaScript-объекты. Их классы заданы в прототипном стиле. В этом легко убедиться, если вывести в консоли любой элемент через console.dir(elem). Или даже можно напрямую обратиться к методам, которые хранятся в Node.prototype, Element.prototype и так далее.

Детальное описание свойств и методов каждого DOM-класса дано в [спецификации](https://html.spec.whatwg.org/multipage/).

Например, [The input element](https://html.spec.whatwg.org/multipage/forms.html#the-input-element) описывает класс, соответствующий <input>, включая [interface HTMLInputElement](https://html.spec.whatwg.org/multipage/forms.html#htmlinputelement), который нас как раз и интересует.

При описании свойств и методов используется не JavaScript, а специальный язык [IDL](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%BE%D0%B2) (Interface Definition Language), который достаточно легко понять «с ходу».

Вот из него выдержка, с комментариями:

|  |
| --- |
| // Объявлен HTMLInputElement  // двоеточие означает, что он наследует от HTMLElement  interface HTMLInputElement: HTMLElement {  // у всех таких элементов есть строковые свойства  // accept, alt, autocomplete, value  attribute DOMString accept;  attribute DOMString alt;  attribute DOMString autocomplete;  attribute DOMString value;  // и логическое свойство autofocus  attribute boolean autofocus;  ...  // а также метод select, который значение не возвращает (void)  void select();  ...  } |

Свойство innerHTML описано в спецификации HTML 5 – [embedded content](http://www.w3.org/TR/html5/embedded-content-0.html).

Оно позволяет получить HTML-содержимое элемента в виде строки. В innerHTML можно и читать и писать.

Пример выведет на экран все содержимое document.body, а затем заменит его на другое:

|  |
| --- |
| <body>  <p>Параграф</p>  <div>Div</div>  <script>  alert( document.body.innerHTML ); // читаем текущее содержимое  document.body.innerHTML = 'Новый BODY!'; // заменяем содержимое  </script>  </body> |

Значение, возвращаемое innerHTML – всегда валидный HTML-код. При записи можно попробовать записать что угодно, но браузер исправит ошибки

# Добавление и удаление узлов

В качестве примера рассмотрим добавление сообщения на страницу, чтобы оно было оформленно красивее чем обычный alert.

HTML-код для сообщения:

|  |
| --- |
| <style>  .alert {  padding: 15px;  border: 1px solid #d6e9c6;  border-radius: 4px;  color: #3c763d;  background-color: #dff0d8;  }  </style>  <div class="alert">  <strong>Ура!</strong> Вы прочитали это важное сообщение.  </div> |

Для создания элементов используются следующие методы:

**document.createElement(tag)**

Создает новый элемент с указанным тегом:

|  |
| --- |
| var div = document.createElement('div'); |

document.createTextNode(text)

Создает новый \*текстовый\* узел с данным текстом:

|  |
| --- |
| var textElem = document.createTextNode('Тут был я'); |

В нашем случае мы хотим сделать DOM-элемент div, дать ему классы и заполнить текстом:

|  |
| --- |
| var div = document.createElement('div');  div.className = "alert alert-success";  div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение."; |

После этого кода у нас есть готовый DOM-элемент. Пока что он присвоен в переменную div, но не виден, так как никак не связан со страницей.

Чтобы DOM-узел был показан на странице, его необходимо вставить в document.

Для этого первым делом нужно решить, куда мы будем его вставлять. Предположим, что мы решили, что вставлять будем в некий элемент parentElem, например var parentElem = document.body.

Для вставки внутрь parentElem есть следующие методы:

**parentElem.appendChild(elem)**

Добавляет elem в конец дочерних элементов parentElem.

Следующий пример добавляет новый элемент в конец <ol>:

|  |
| --- |
| <ol id="list">  <li>0</li>  <li>1</li>  <li>2</li>  </ol>  <script>  var newLi = document.createElement('li');  newLi.innerHTML = 'Привет, мир!';  list.appendChild(newLi);  </script> |

**parentElem.insertBefore(elem, nextSibling)**

Вставляет elem в коллекцию детей parentElem, перед элементом nextSibling.

Следующий код вставляет новый элемент перед вторым <li>:

|  |
| --- |
| <ol id="list">  <li>0</li>  <li>1</li>  <li>2</li>  </ol>  <script>  var newLi = document.createElement('li');  newLi.innerHTML = 'Привет, мир!';  list.insertBefore(newLi, list.children[1]);  </script> |

Для удаления узла есть два метода:

**parentElem.removeChild(elem)**

Удаляет elem из списка детей parentElem.

**parentElem.replaceChild(newElem, elem)**

Среди детей parentElem удаляет elem и вставляет на его место newElem.

Оба этих метода возвращают удаленный узел, то есть elem. Если нужно, его можно вставить в другое место DOM тут же или в будущем.

|  |
| --- |
| <style>  .alert {  padding: 15px;  border: 1px solid #d6e9c6;  border-radius: 4px;  color: #3c763d;  background-color: #dff0d8;  }  </style>  <body>  <h3>Сообщение пропадёт через секунду</h3>  </body>  <script>  var div = document.createElement('div');  div.className = "alert alert-success";  div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";  document.body.appendChild(div);  setTimeout(function() {  div.parentNode.removeChild(div);  }, 1000);  </script> |

# Домашнее задание

* Создать функцию, генерирующую шахматную доску. При этом можно использовать любые html-тэги по своему желанию. Доска должна быть разленована соответствующим образом, т.е. чередовать черные и белые ячейки. Строки должны нумероваться числами от 1 до 8, столбцы – латинскими буквами A, B, C, D, E, F, G, H.
* Заполнить созданную таблицу буквами, отвечающими за шахматную фигуру, например К-король, Ф – ферзь и тп., причем все фигуры должны стоять на своих местах и быть соответственно черными и белыми.
* \* Заменить буквы, обозначающие фигуры картинками.

# Практика

Продолжаем разрабатывать адресную книгу. В этом уроке мы сделаем вывод пользователей на страницу. И добавим изменение страницы при вызове методов добавления и удаления контакта. Начнем с методов генерации разметки для списка пользователей. Методов у нас будет 2. Первый будет заниматься генерацией разметки для одного контакта:

**function** printUser(user){

**var** row = document.createElement('div');

row.id = 'u\_' + user.id;

row.className = 'row';

row.innerHTML = '<div class="col">' + user.firstName + '</div>' + '<div class="col">' + user.lastName + '</div>' +

'<div class="col">' + user.birthDay + '</div>' + '<div class="col">' + user.phones.**join**(', ') + '</div>' + '<div class="del\_btn" onclick="delHandler(event)">X</div>';

**return** row;

}

А второй, вызывая первый метод, будет генерировать разметку для списка контактов:

window.onload = **function**(event) {

**var** usersTable = document.getElementById('users');

users.forEach(**function**(user){

usersTable.appendChild(printUser(user));

});

};

Конструкция window.onload означает, что указанный код должен запуститься только после того, как страница будет полностью загружена. Более подробно мы с Вами о таких конструкциях поговорим на следующем занятии.

Вывод списка пользователей у нас готов. Теперь модифицируем функции удаления и добавления контактов таким образом, чтобы при их вызове отображение страницы тоже менялось. Для этого достаточно очищать разметку и генерировать ее снова в конце вызова соответствующего метода. Данная задача остается Вам на самостоятельную реализацию.

# Используемая литература

* [Mozilla Developer Network](https://developer.mozilla.org/)
* [MSDN](http://msdn.microsoft.com/)
* [Safari Developer Library](https://developer.apple.com/library/safari/navigation/index.html)
* [Современный учебник JavaScript](http://learn.javascript.ru)