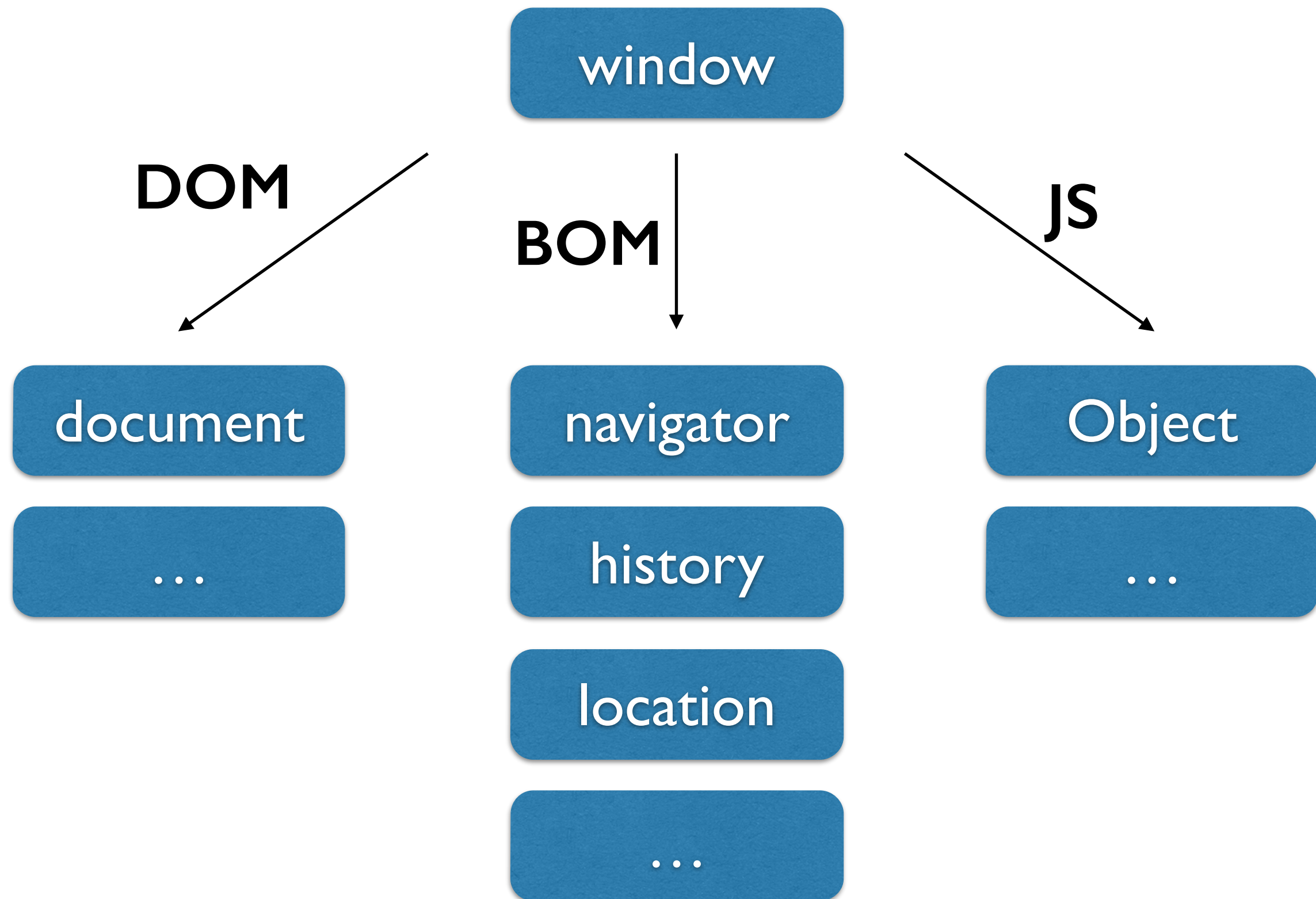


Front-end

DOM / BOM

DOM, BOM, JS



BOM

BOM = Browser Object Model

— объектная модель браузера

Не существует официального стандарта

JavaScript всегда выполняется в рамках глобального объекта.

В браузере это **window** (Node JS – `global`).

— `window` — окно, содержащее документ

— глобальный объект

— окно для текущего документа можно получить с помощью **`document.defaultView`**

— каждый таб — отдельный `window`

Для обращения к функциям и свойствам `window` не нужно указывать объект:

```
window.setInterval(..)
```

// то же что и

```
setInterval(..)
```

Любая переменная, если не найдена локально, в конечном итоге ищется в глобальном объекте.

Объекты BOM

navigator

Содержит общую информацию о браузере и системе

navigator.userAgent; // информация о браузере, например: Mozilla/5.0
(Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/33.0.1750.152 Safari/537.36

navigator.platform; // информация о платформе, например: MacIntel

screen

Содержит общую информацию об экране (разрешение, цветность и т.д.)

screen.width; // 1920
screen.height; // 1200

history

Позволяет менять адрес без перезагрузки страницы (в пределах того же домена) при помощи History API, а также перенаправлять посетителя назад-вперед по истории.

history.back();
history.forward();

Объекты BOM

location

Предоставляет информацию о текущем URL

```
location.toString();
```

// вернет полный адрес, например "http://www.w3schools.com/default.asp"

http://www.google.com:80/search?q=javascript#test

Свойство	Описание	Пример
hash	часть URL, которая идет после символа решетки '#', включая символ '#'	#test
host	хост и порт	www.google.com:80
href	весь URL	http://www.google.com:80/search?q=javascript#test
hostname	хост (без порта)	www.google.com
pathname	строка пути (относительно хоста)	/search
port	номер порта (если порт не указан, то пустая строка)	80
protocol	протокол	http: (двоеточие на конце)
search	часть адреса после символа "?", включая символ"?"	?q=javascript

Объекты BOM

Методы объекта location

`location.assign(url);` равносильно `location.href = url;`

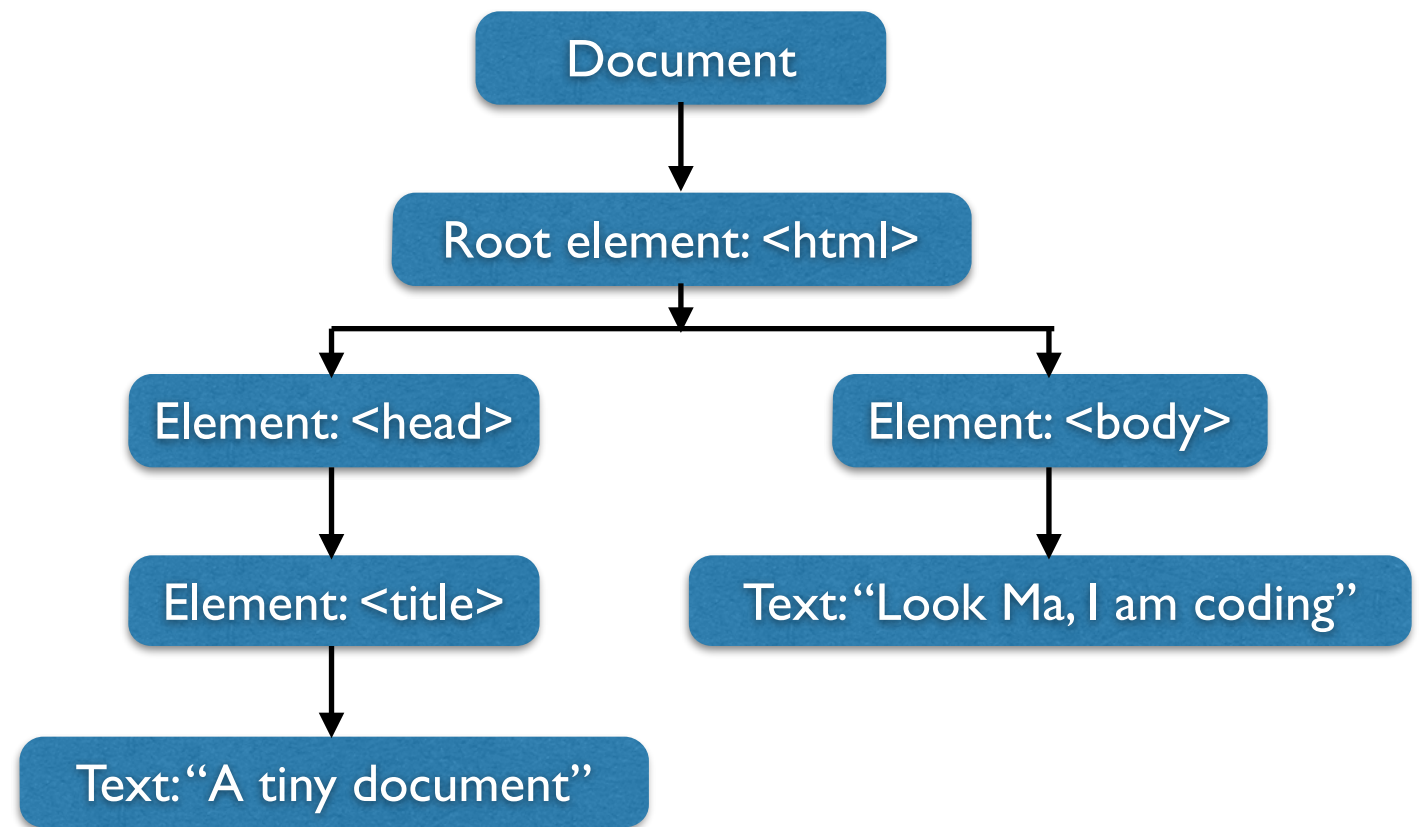
`location.replace(url);`

// перезапишет историю, невозможно будет вернуться с помощью кнопки «Назад» на страницу, с которой ушли

`location = url;`

DOM

```
<html>
<head>
  <title>A tiny document</title>
</head>
<body>
  Look Ma, I am coding
</body>
</html>
```



DOM = Document Object Model – представление веб-страницы в виде упорядоченной группы узлов и объектов, имеющих свойства и методы (дерево)

- интерфейс программирования для HTML и XML документов
- не зависит от языка
- не зависит от платформы
- стандарт W3C
- браузеры используют разную имплементацию DOM, многие из них предлагают расширения над стандартом

DOM + JavaScript

С помощью JavaScript, используя DOM, можно получить доступ к документу и его элементам.

Все элементы являются частью объектной модели документа => можно получить доступ к любому элементу: head, таблицы, текст, заголовки и т.д.

API (веб страница / XML) = DOM (содержимое страницы) + JS (scripting language)

- Изменять/удалять/добавлять элементы страницы
- Изменять/удалять/добавлять атрибуты элементов
- Изменять/удалять/добавлять стили
- Обработать события, происходящие на странице
- Создавать новые события
- Передвигаться по дереву

Работа с DOM из консоли

Chrome -> правой кнопкой мыши по элементу -> Inspect Element => Developer Tools

The screenshot shows the Chrome Developer Tools interface. The 'Elements' pane on the left displays the DOM tree, with the element `<div title="Google" align="left" id="hplogo" onload="window.lol&&lol()" style="background:url(/images/srpr/logo11w.png) repeat;background-size:269px 95px;height:95px;width:269px">` selected. The 'Styles' pane on the right shows the styles for the selected element, including `background: url(/images/srpr/logo11w.png) no-repeat;`, `background-size: 269px 95px;`, `height: 95px;`, and `width: 269px;`. The breadcrumb at the bottom indicates the path: `#gsr #viewport #main span#body.ctr-p center div#lga div div#hplogo`.

```
</div>
</div>
</div>
<div id="gbw"></div>
</div>
<div id="gba"></div>
</div>
▶<div class="s2fp-h spch" id="spch">...</div>
▼<div class="content" id="main">
  ▼<span class="ctr-p" data-jiis="bp" id="body">
    ▼<center>
      ▼<div id="lga" style="height:231px;margin-top:20px">
        ▼<div style="padding-top:112px">
          ▶<div title="Google" align="left" id="hplogo" onload=
            "window.lol&&lol()" style="background:url(/images/srpr/logo1
            repeat;background-size:269px 95px;height:95px;width:269px">...
          </div>
        </div>
        <div style="height:102px"></div>
        ▶<div id="prm-pt" style="margin-top:12px">...</div>
      </center>
    </span>
    ▶<div class="ctr-p" data-jiis="bp" id="footer">...</div>
  </div>
  <script>...</script>
  <script data-url="/extern_chrome/f2c7ac76c5684519.js?bav=or.r_qf" i
    "ecs"></script>
</div>
#gsr #viewport #main span#body.ctr-p center div#lga div div#hplogo
```

Styles

```
element.style {
  background: url(/images/srpr/logo11w.png) no-repeat;
  background-size: 269px 95px;
  height: 95px;
  width: 269px;
}

div[Attributes Style] {
  text-align: -webkit-left;
}

div {
  display: block;
}
user agent stylesheet

Inherited from center
center {
  text-align: -webkit-center;
}
user agent stylesheet

Inherited from body#gsr.hp.vasq
body, html {
  font-size: small;
}
?qws_rd=cr&ei=p...l4QSYqIHqAq:10

body {
  color: #222;
}
?qws_rd=cr&ei=p...l4QSYqIHqAq:10
```

Работа с DOM из консоли

Firefox -> правой кнопкой мыши по элементу -> Inspect Element => Web Developer Tools

The screenshot displays the Firefox Web Developer Tools interface. The top toolbar includes buttons for Console, Inspector, Debugger, Style Editor, Profiler, and Network. The Inspector panel is active, showing a breadcrumb trail: `page.c` > `div.amo-header` > `div#masthead` > `h1.site-title` > `a`. The DOM tree on the left shows the HTML structure, with the selected `a` element highlighted. The CSS rules pane on the right lists several styles, including `element { inline}`, `.site-title a:hover`, `.site-title a:focus`, `.site-title a {`, `a {`, and a base selector `html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label,`. The selected `a` element has the `color` property set to `#4478C4` and `text-decoration` set to `none`.

```
<!DOCTYPE html>
<html lang="en-US" dir="ltr">
  <head></head>
  <body class="html-ltr firefox moz-header-slim gutter addon-details
is-im...recaptcha_nothad_incorrect_sol recaptcha_isnot_showing_audio" data-collect-
timings="/services/timing/record:0.1" data-media-url="https://addons.cdn.mozilla.net
/media/" data-readonly="false" data-anonymous="true" data-nightly-version="28.0"
data-min-beta-version="3.7" data-appid="1" data-appname="Firefox" data-app="firefox">
    <div id="tabzilla-panel" class="tabzilla-closed" tabindex="-1"></div>
    <div id="tabzilla-wrapper">
      <div id="page" class="c">
        <div id="global-header-tab"></div>
        <div class="amo-header">
          <nav id="aux-nav" class="menu-nav c" role="navigation"></nav>
          <div class="header-search" role="search"></div>
          <div id="masthead">
            <h1 class="site-title">
              <a title="Return to the Firefox Add-ons homepage" href="/en-US/firefox/"
              ></a>
            </h1>
            <nav id="site-nav" class="menu-nav c"></nav>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

element { inline

.site-title a:hover, impala-min.css:1

.site-title a:focus {

color: #043B84;

text-shadow: 0px 0px 40px #FFF, 0px 0px 20px #FFF, 0px 0px 10px #FFF;

.site-title a { impala-min.css:1

color: #333;

text-decoration: none;

a { impala-min.css:1

text-decoration: none;

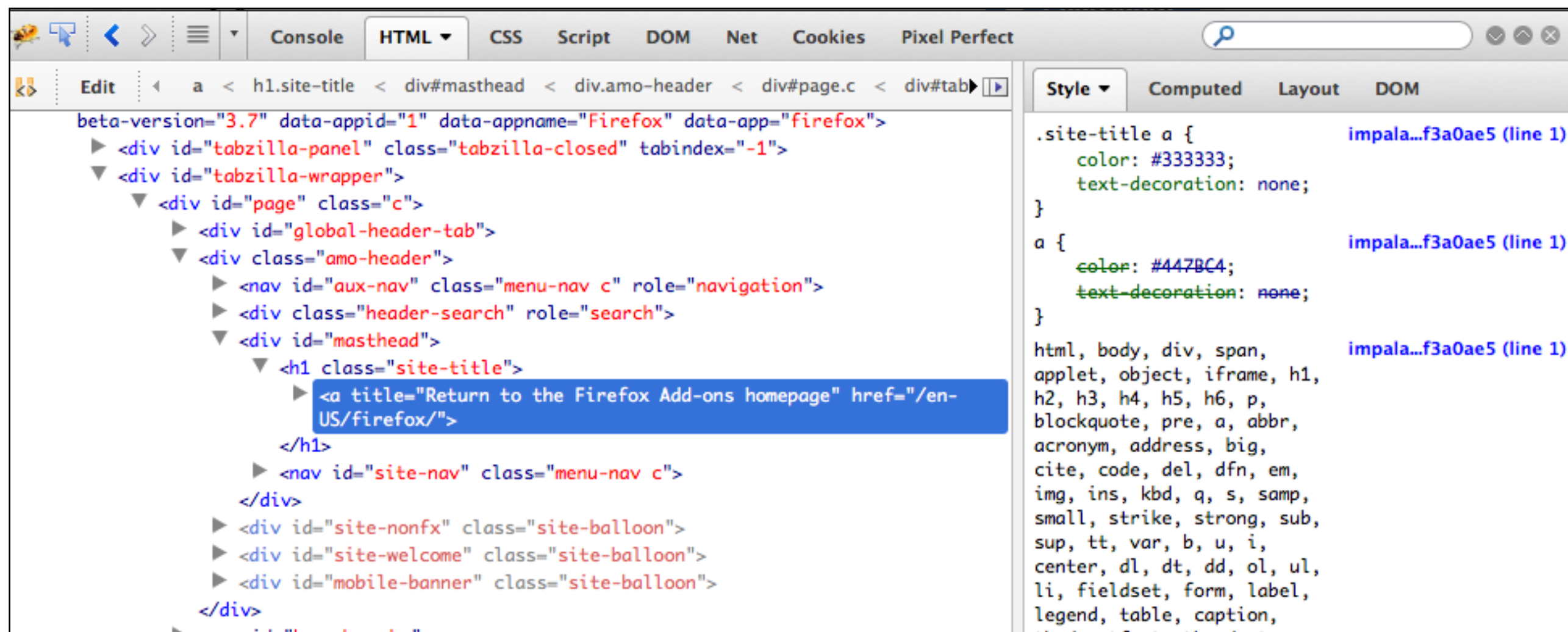
color: #4478C4;

html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, img, ins, kbd, q, s, samp, small, strike, strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label,

Работа с DOM из консоли

Firefox

Плагин Firebug: <https://addons.mozilla.org/en-US/firefox/addon/firebug/>

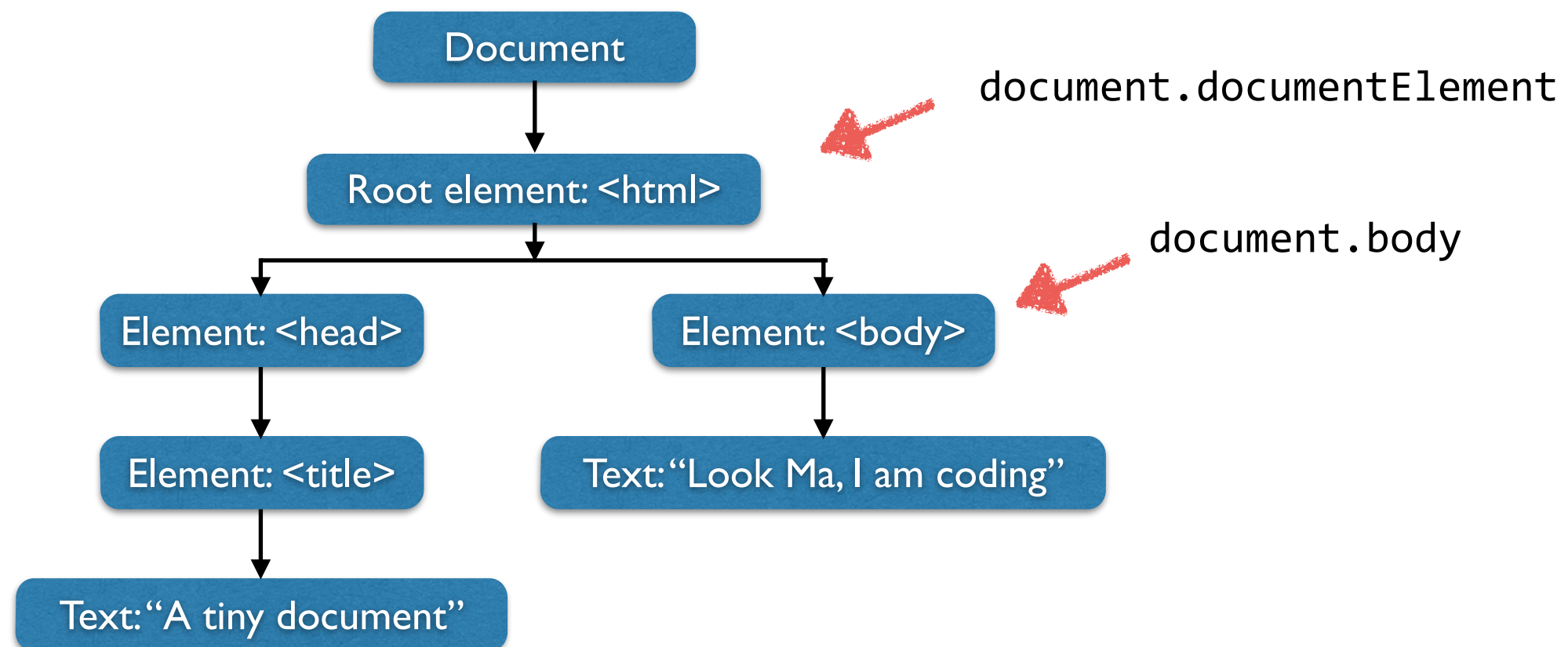


Доступ к DOM

Доступ к DOM начинается с объекта **document**.

document.documentElement ссылается на DOM-объект для тега `<html>`

document.body соответствует тегу `<body>`



Доступ к DOM



Нельзя получить доступ к элементу, которого еще не существует в момент выполнения скрипта


```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      alert(document.body); // null
    </script>
  </head>
  <body>
    <script>
      alert(document.body); // [object HTMLBodyElement]
    </script>
  </body>
</html>
```


Навигация в DOM

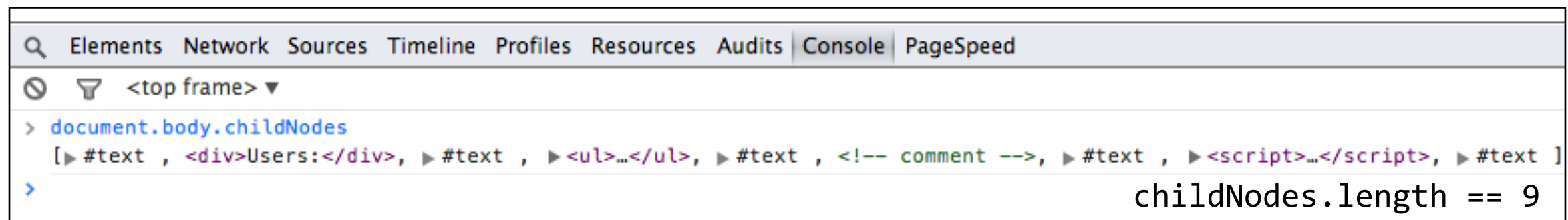
Из узла-родителя можно
получить все дочерние
элементы.

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

```
<!DOCTYPE html>
<html>
  <head> <title></title> </head>
  <body>
    <div>Users:</div>
    <ul>
      <li>Paul</li>
      <li>Jill</li>
    </ul>
    <!-- comment -->
    <script>
      var childNodes = document.body.childNodes;
      console.log(childNodes.length);
    </script>
  </body>
</html>
```



8



Навигация в DOM

children перечисляет только дочерние узлы, соответствующие тегам.

```
<!DOCTYPE html>
<html>
  <head> <title></title> </head>
  <body>
    <div>Users:</div>
    <ul>
      <li>Paul</li>
      <li>Jill</li>
    </ul>
    <!-- comment -->
    <script>
      var children = document.body.children;
      alert(children.length);
    </script>
  </body>
</html>
```

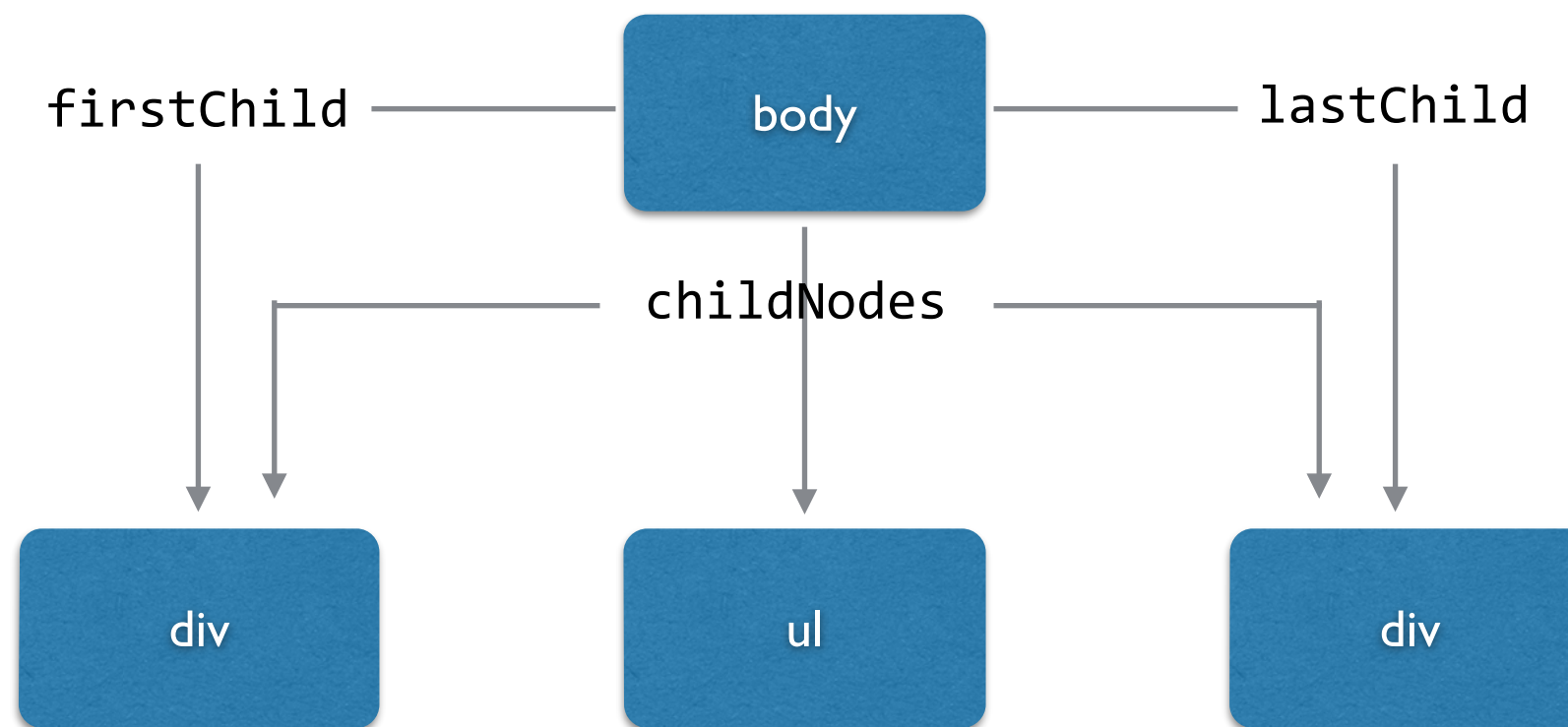
← 3
?



Навигация в DOM

Свойства **firstChild** и **lastChild** обеспечивают быстрый доступ к первому и последнему потомку.

```
<html><body><div>...</div><ul>...</ul><div>...</div></body></html>
```



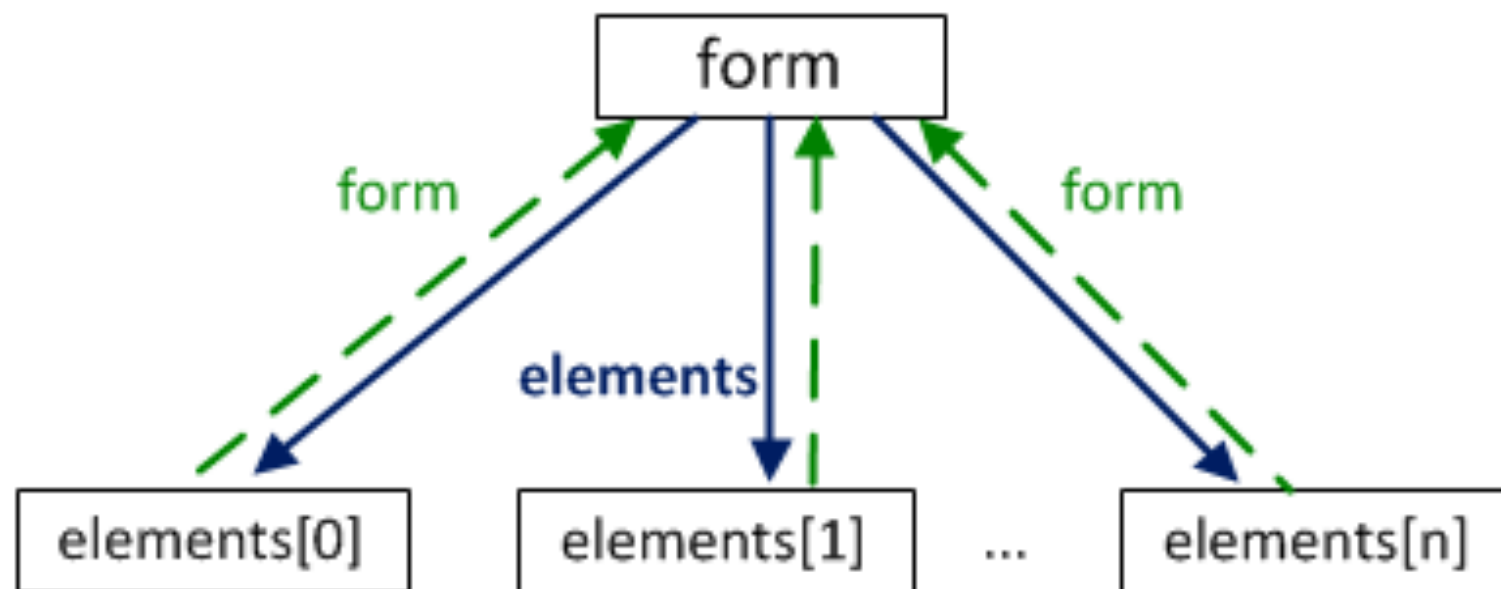
Формы

Форму можно получить по имени либо индексу.

`document.forms.my` – форма с именем 'my'

`document.forms[0]` – первая форма в документе

Любой элемент формы `form` можно получить аналогичным образом, используя свойство `form.elements`



Может быть несколько элементов с одинаковым именем. В таком случае `form.elements[name]` вернет коллекцию элементов.

`element.form` вернет форму

Таблицы

У таблиц есть дополнительные свойства для более удобной навигации по ним. Некоторые из них:

table.rows — список строк (tr) таблицы.

table.caption/tHead/tFoot — ссылки на элементы таблицы caption, thead, tfoot.

table.tBodies — список элементов таблицы tbody, по спецификации их может быть несколько.

tr.cells — список ячеек (td) таблицы.

td.cellIndex — номер ячейки в строке

```
<table id="content">
  <tr>
    <td>One</td><td>Two</td>
  </tr>
  <tr>
    <td>Three</td><td>Four</td>
  </tr>
</table>
```

```
<script>
  var table = document.getElementById("content");
  console.log(table.rows[0].cells[0].textContent); // One
</script>
```

Свойства узлов

innerHTML

Получает либо задает содержимое в виде HTML

```
content = element.innerHTML;  
element.innerHTML = content;
```

```
<div id="content">Some text here</div>
```

```
// 1  
content =  
document.getElementById("content");  
content.innerHTML = '<p>test</p>';  
// Результат:  
<div id="content"><p>test</p></div>
```

```
// 3  
content.innerHTML = 'abc&';  
console.log(content.innerHTML);  
// abc&amp;  
console.log(content.textContent);  
// abc&;
```

```
// 2  
content.innerHTML = '';  
// Результат:  
<div id="content"></div>
```

```
! // 4  
content.innerHTML =  
'<script>alert(1);</scr'+ 'ipt>';  
// не выполнится  
content.innerHTML =  
'<img src=x onerror=alert(1)>';  
// выполнится!
```

Для вставки простого текста корректнее пользоваться свойством `textContent`

Свойства узлов

У DOM-узлов есть свойства, общие для всех элементов (свойства `HTMLElement`):

id – идентификатор

tagName – название элемента (“span”)

и другие

Также есть свойства, которые зависят от типа элемента:

href – адрес ссылки

value – значение для `input`, `select`, `textarea`

type – тип поля ввода

name – имя элемента, применимо к `a`, `button`, `input`, `img`, `form`, `texture`, `select` ...

selectedIndex – индекс выбранного значения для `select`

и многие другие

Стандартные свойства DOM синхронизируются с атрибутами:

``

`document.getElementById("a").href`

`<input id="b" type="checkbox" checked>`

`document.getElementById("b").checked`

`<input id="c" type="text" value="markup">`

`document.getElementById("c").value`

Атрибуты

setAttribute

Задаёт либо изменяет существующий атрибут элемента

```
element.setAttribute(name, value);
```

name - имя атрибута

value - значение атрибута

```
var d = document.getElementById("content");  
d.setAttribute("align", "center");
```

getAttribute

Возвращает значение атрибута по его имени либо null, если атрибут не задан (или "")

```
element.getAttribute(name);
```

name - имя атрибута

```
var d = document.getElementById("content");  
d.getAttribute("align");
```

Атрибуты

hasAttribute

Возвращает true/false в зависимости от того, присутствует ли данный атрибут или нет

element.hasAttribute(name);

name - имя атрибута

```
var d = document.getElementById("content");  
console.log(d.hasAttribute("align"));
```

removeAttribute

Удаляет у элемента указанный атрибут

element.removeAttribute(name);

name - имя атрибута

```
var d = document.getElementById("content");  
d.removeAttribute("align");
```

! Следует использовать **removeAttribute(name)** вместо **element.setAttribute(name, null);**

При попытке удалить несуществующий атрибут, ошибки не возникнет

Добавление узлов

createElement

Возвращает элемент указанного типа либо HTMLUnknownElement, если элемент не известен

```
element = document.createElement(tagName);
```

tagName — имя тэга

```
<div id="content">Some text here</div>
```

```
var content = null,  
    paragraph = null,  
    text = null;
```

```
paragraph = document.createElement("p");  
text = document.createTextNode("Hi there!");
```

```
paragraph.appendChild(text);
```

```
content = document.getElementById("content");  
document.body.insertBefore(paragraph, content);
```

```
<p>Hi there!</p>
```

```
<div id="content">Some text here</div>
```


Добавление узлов

appendChild

Добавляет элементу дочерний узел, помещает его самым последним

child = **element.appendChild(child)**; // возвращает его же

element — родительский элемент

child — добавляемый элемент типа Node

```
var p = document.createElement("p");  
document.body.appendChild(p);
```

Если **child** — ссылка на уже существующий элемент в документе, то этот элемент перемещается с текущей позиции в новую

Один и тот же узел не может находиться в нескольких местах документа одновременно

Поиск элементов

getElementById

Возвращает ссылку на элемент в дереве по его уникальному ID

```
element = document.getElementById(id);  
// объект типа Element либо null
```

```
var d = document.getElementById("content");  
d.id; // "content"
```

Параметр ID чувствителен к регистру

```
document.getElementById("Content"); // null
```

getElementsByClassName

Возвращает массив дочерних документов с заданным CSS классом

```
elements = document.getElementsByClassName(names);  
// объект типа HTMLCollection
```

```
var d = document.getElementsByClassName("red link test");  
document.getElementById("content").getElementsByClassName("yellow");
```

Поиск элементов

getElementsByTagName

Возвращает список элементов по указанному имени тэга

elements = element.getElementsByTagName(tagName);

elements — список типа NodeList (HTMLCollection) элементов в том порядке, в котором они расположены в дереве (*live* = обновляется автоматически вместе с DOM деревом), либо пустой список, если элементы не найдены

element — элемент, с которого необходимо начать поиск. Сам элемент в результаты поиска не включается, только его потомки

tagName — имя тэга, * — все тэги

```
var table = document.getElementById("forecast-table");
var cells = table.getElementsByTagName("td");
for (var i = 0; i < cells.length; i++) {
    var status = cells[i].getAttribute("data-status");
    if ( status == "open" ) {
        // grab the data
    }
}
```

Внешний вид элементов

style, getComputedStyle

Свойство объекта HTMLElement, представляющее собой атрибут **style** у элемента

```
<p id="content"
  style="color:red; margin:10px 5px; font-weight:bold;" class="wrapper">
  Some text here
</p>
```

```
var d = document.getElementById("content");
d.style.color = "blue"; // изменит только цвет текста
```

```
var cs = window.getComputedStyle(content, null); // объект CSSStyleDeclaration
// вернет значения всех CSS свойств (установленные вручную, стили по умолчанию)
```

```
d.setAttribute('style', 'color: blue'); // перезапишет все стили
```

className

```
d.className; // wrapper
d.className = "clearfix";
```

События

Для реакции на действия посетителя и внутреннего взаимодействия скриптов существуют события.

Событие - это сигнал от браузера о том, что что-то произошло

Существует много видов событий.

DOM-события, которые инициализируются элементами DOM.

Например:

Событие **click** происходит, когда кликнули на элемент

Событие **mouseover** — когда на элемент наводится мышь.

Событие **focus** — когда посетитель фокусируется на элементе.

Событие **keydown** — когда посетитель нажимает клавишу.

События для окна браузера.

Например, **resize** — когда изменяется размер окна.

События загрузки файла/документа.

load,readystatechange, DOMContentLoaded...

Про события хорошо написано здесь: <http://learn.javascript.ru/events>

События

Необходимо помнить, что нельзя получить доступ к элементу, которого еще не существует в момент выполнения скрипта

Как гарантировать, что в момент выполнения скрипта, необходимый нам элемент уже был загружен в DOM-дерево?

- 1) поместить скрипт в конец документа, прямо перед закрывающим тэгом `</body>`
- 2) использовать специальные события, указывающие на загрузку содержимого страницы

DOMContentLoaded: происходит, когда документ был полностью загружен и обработан, не дожидаясь загрузки стилей и изображений

load: можно использовать для определения полностью загруженной страницы. Срабатывает, когда ресурс и все вложенные ресурсы были загружены

```
<body onload="init()">
```

```
...
```

```
function init() {
```

```
  ...
```

```
}
```

Событийные модели

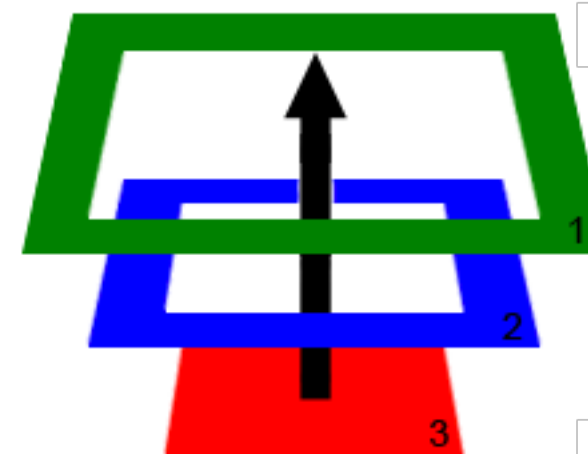
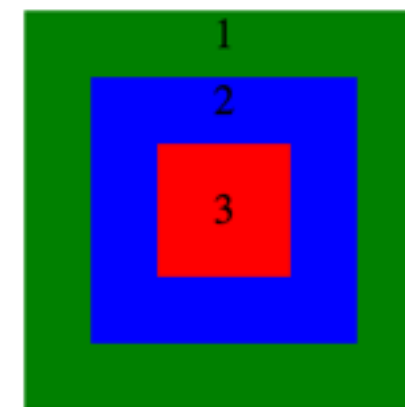
Элементы DOM могут быть вложены друг в друга. При этом обработчик, привязанный к родителю, срабатывает, даже если посетитель кликнул по потомку.

Это происходит потому, что событие всплывает.

Всплытие

```
<div class="d1">1 <!-- topmost -->
  <div class="d2">2
    <div class="d3">3 <!-- innermost -->
  </div>
</div>
```

После того, как событие сработает на самом вложенном элементе, оно также сработает на родителях, вверх по цепочке вложенности.



The topmost element

The innermost element

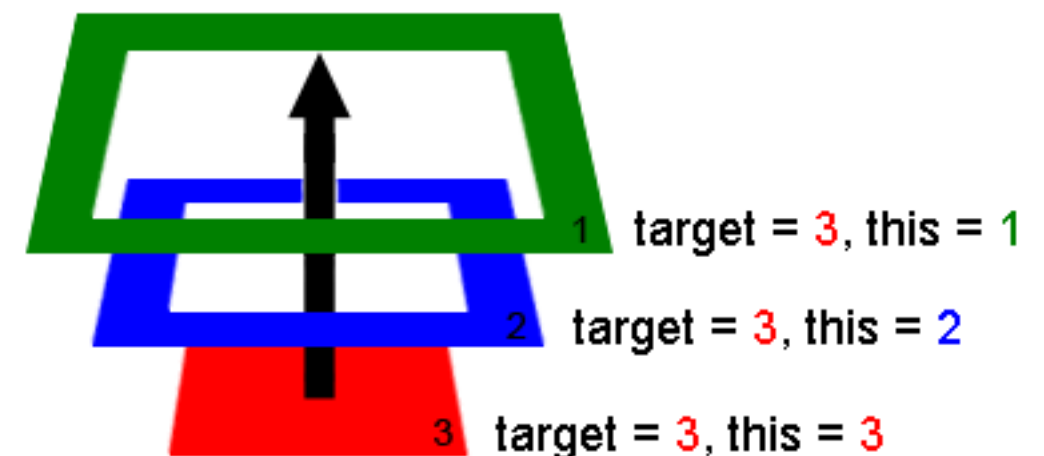
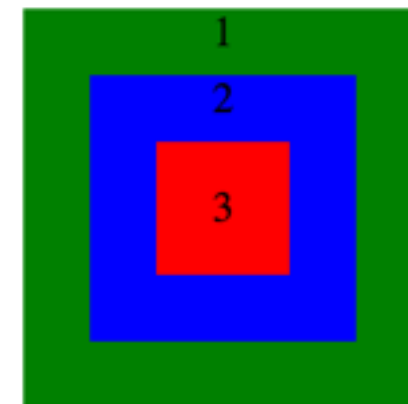
Событийные модели

Необходимо различать 2 понятия:

- 1) целевой элемент, самый глубокий, тот который вызывает событие – **event.target**
- 2) элемент, на котором сработал обработчик – **this**

```
<div class="d1">1 <!-- topmost -->
  <div class="d2">2
    <div class="d3">3 <!--innermost -->
    </div>
  </div>
</div>
```

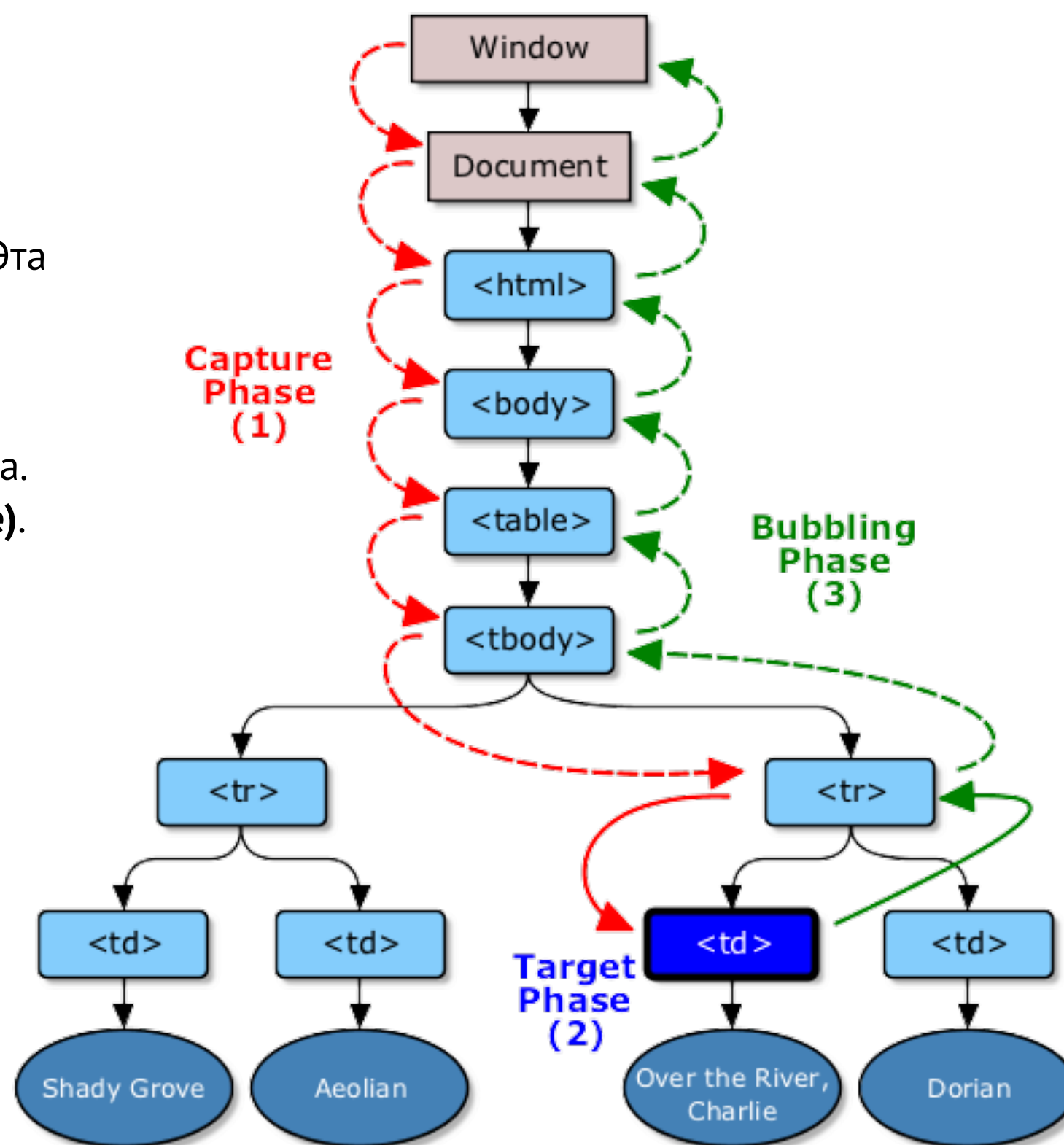
```
for(var i = 0; i < divs.length; i++) {
  divs[i].onclick = function(e) {
    alert(event.target.className);
    alert(this.className);
  }
}
```



Стадии прохода события

Во всех браузерах, кроме IE<9, есть три стадии прохода события.

- 1) Событие сначала идет сверху вниз. Эта стадия называется **«стадия перехвата» (capturing stage)**.
- 2) Событие достигло целевого элемента. Это — **«стадия цели» (target stage)**.
- 3) После этого событие начинает всплывать. Это — **«стадия всплытия» (bubbling stage)**.



Назначение обработчиков событий

1) `<input id="b1" value="Click" onclick="alert('Clicked!')" type="button"/>`

2) `<input id="myElement" type="button" value="Click"/>`
`<script>`
 `var elem = document.getElementById('myElement');`
 `elem.onclick = function(event) {`
 `alert('Clicked!');`
 `}`
`</script>`

Недостаток обоих способов — нельзя назначить больше одного обработчика:

```
<input type="button" onclick="alert('Before')" value="Click"/>
<script>
    var elem = document.getElementsByTagName('input')[0];
    elem.onclick = function(event) {
        alert('After');
    }
</script>
```

! `function sayHi(event) {`
 `alert('hi!');`
 `}`
`document.getElementById('button').onclick = sayHi;`

Назначение обработчиков событий

addEventListener (все браузеры, кроме IE<9)

Регистрирует указанный обработчик события

```
element.addEventListener(type, listener[, useCapture]);
```

element – объекты Element, document, window

type – тип события (строка) (click, mouseover, ...)

listener – функция-обработчик события

useCapture (опционально, **false** по умолчанию) – позволяет задать стадию, на которой будет поймано событие.

Если аргумент true, то событие будет перехвачено по дороге вниз (перехват).

Если аргумент false, то событие будет поймано при всплытии.

Назначение обработчиков событий

addEventListener (все браузеры, кроме IE<9)

```
<div id="content">  
  <p>First paragraph</p>  
  <p>Second paragraph</p>  
</div>
```

```
function modifyText() {  
  var content = document.getElementById("content"),  
      paragraph = content.getElementsByTagName('p')[0];  
  paragraph.firstChild.nodeValue = "New text";  
}
```

```
var el = document.getElementById("content");  
el.addEventListener("click", modifyText);
```

Назначение обработчиков событий

removeEventListener (все браузеры, кроме IE<9)

Удаляет ранее добавленный с помощью метода addEventListener обработчик события

element.removeEventListener(type, listener[, useCapture]);

element – объекты Element, document, window (на котором был назначен обработчик)

type – тип события (строка)

listener – функция-обработчик события

useCapture (опционально, **false** по умолчанию) – указывает, был ли обработчик задан как перехватывающий (true) или как обработчик на стадии всплытия (false)

```
var div = document.getElementById('div');
var listener = function (event) {
  /* do something here */
};
div.addEventListener('click', listener);
div.removeEventListener('click', listener);
```

Итого

- 1) Браузер дает доступ к иерархии объектов, которые мы можем использовать для разработки. JavaScript служит нам инструментом:
 - 1) DOM дает доступ к содержимому страницы
 - 2) BOM дает возможность работать с окружением документа (браузером): передвигаться по истории, получать информацию о браузере и системе пользователя
- 2) Используя DOM, можно получить доступ к элементам страницы.
 - 1) Изменять/удалять/добавлять элементы страницы
 - 2) Изменять/удалять/добавлять атрибуты элементов
 - 3) Изменять/удалять/добавлять стили
 - 4) Обработать события, происходящие на странице или создавать новые
 - 5) Передвигаться по дереву
- 3) Существует несколько способов поиска элементов в дереве
- 4) Существует несколько способов навигации в дереве
- 5) Существует множество свойств у объектов (общие для всех, специальные для разных типов); стандартные свойства синхронизируются с атрибутами
- 6) Существует несколько способов обработать события
- 7) Стадии прохода события

The End

Exadel[®]