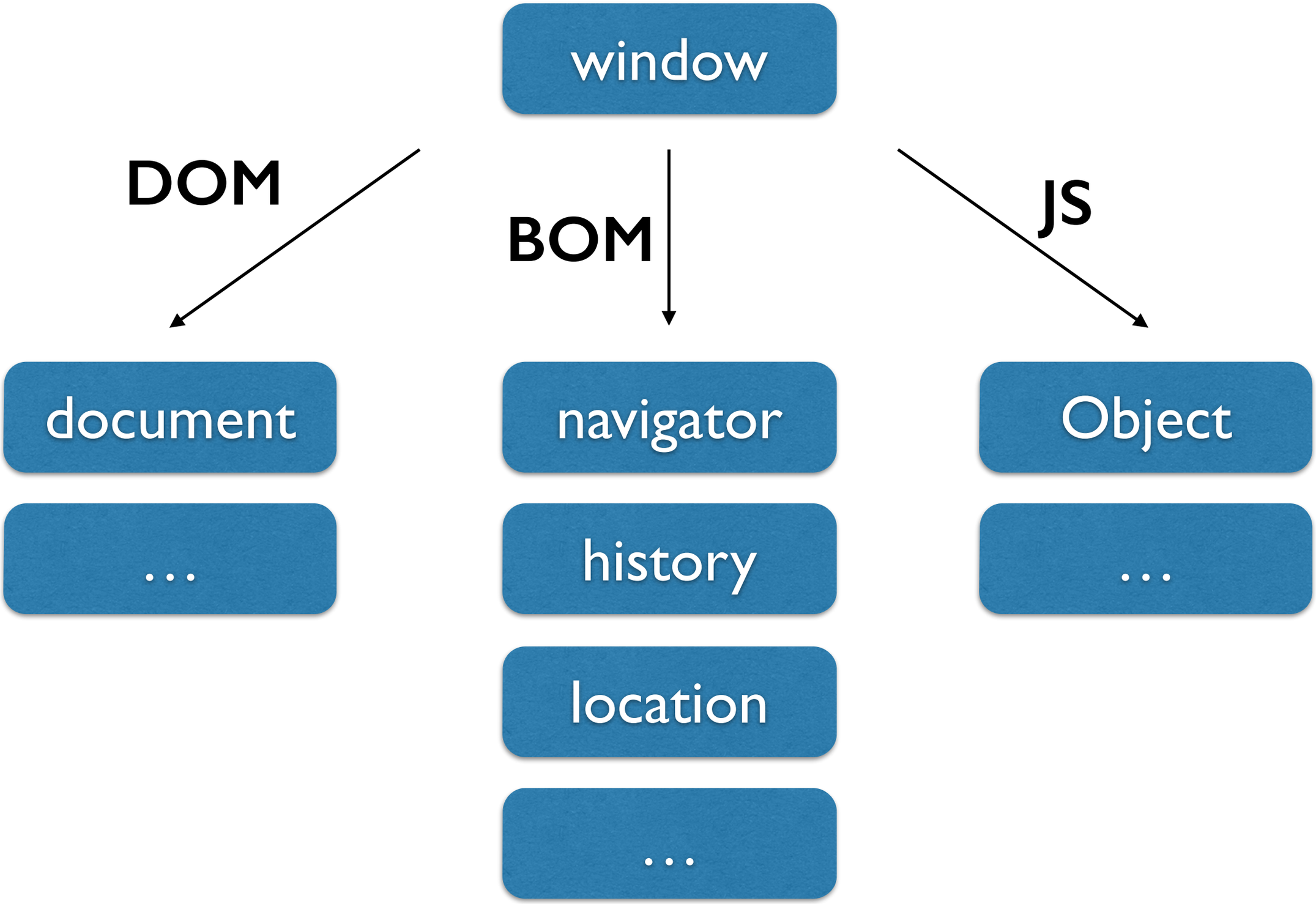
Front-end

DOM / BOM

Exadel®

DOM, BOM, JS


window
DOM
BOM
JS
document
…
navigator
history
location
…
Object
…

BOM

BOM = Browser Object Model (объектная модель браузера)

- Не существует официального стандарта
- JavaScript всегда выполняется в рамках глобального объекта. В браузере это **`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

BOM

- `window` —
  - окно, содержащее документ
  - глобальный объект

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

- каждый таб — отдельный `window`

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
```

Объекты BOM

screen

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

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

Объекты BOM

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

location

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

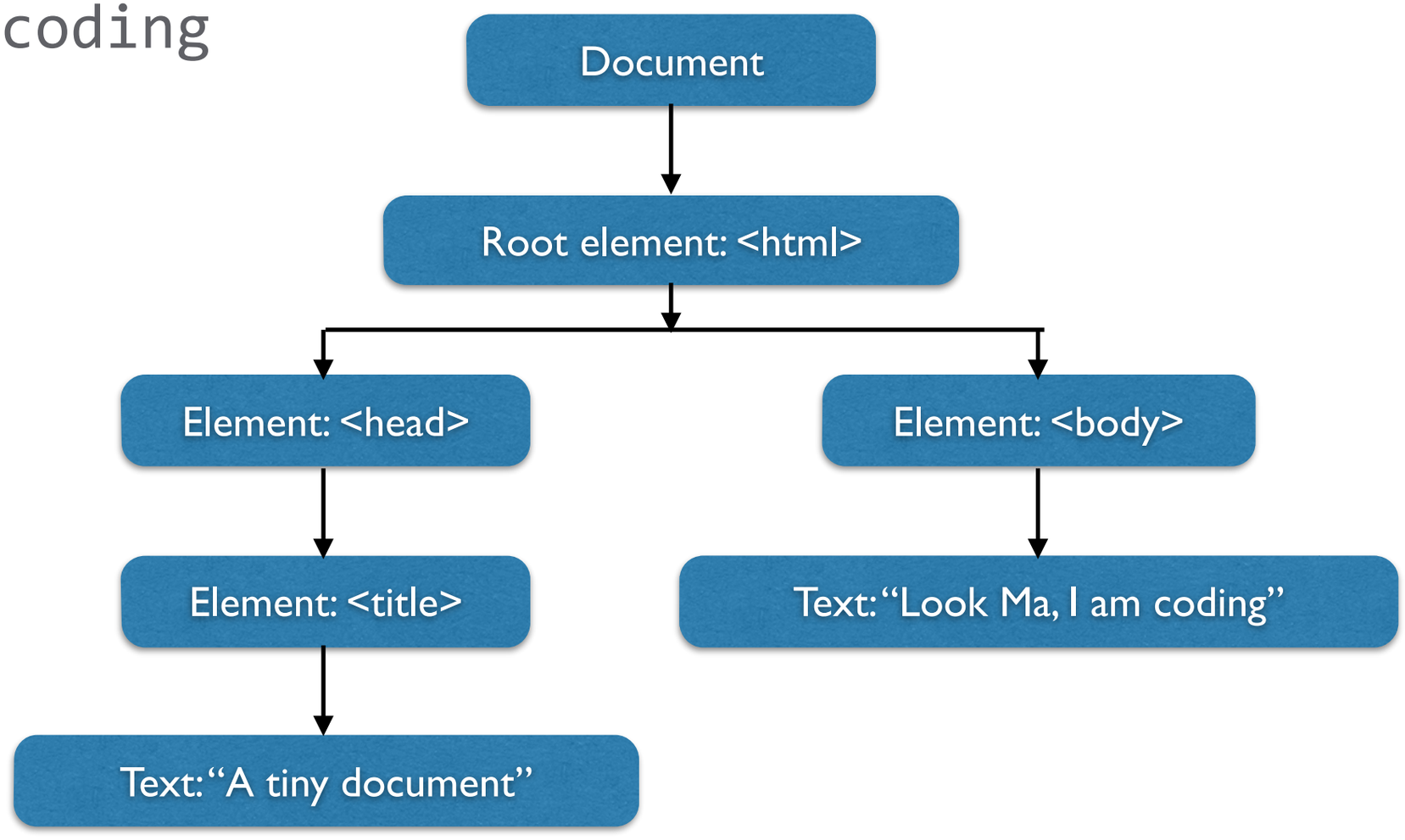
```
location.replace(url);
```

// 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

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

API (веб страница / XML)

=

DOM (содержимое страницы)

+

JS (scripting language)

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.

Какие есть возможности?

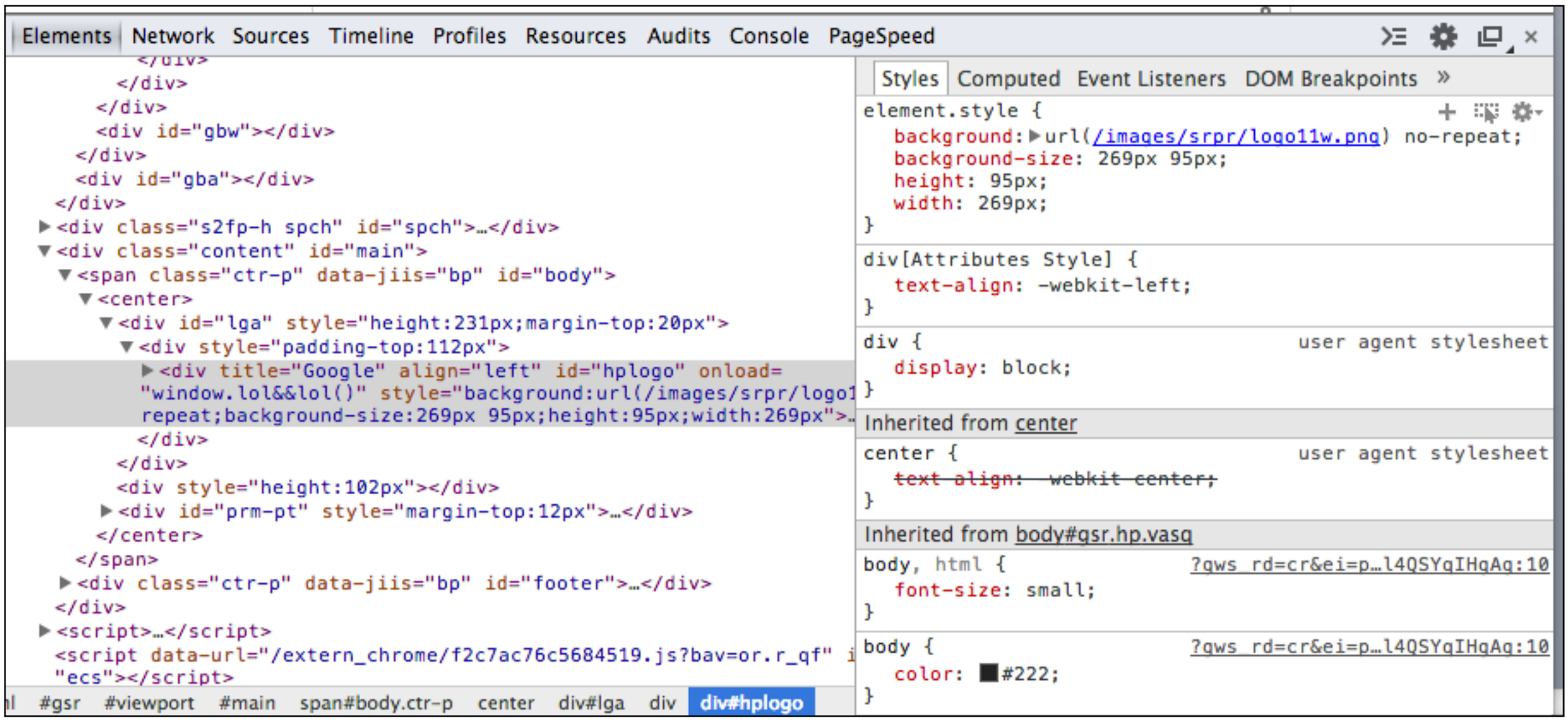
- Изменять/удалять/добавлять элементы страницы
- Изменять/удалять/добавлять атрибуты элементов
- Изменять/удалять/добавлять стили
- Обрабатывать события, происходящие на странице
- Создавать новые события
- Передвигаться по дереву

Работа с DOM из консоли

Chrome

-> правой кнопкой мыши по элементу

-> Inspect Element => Developer Tools

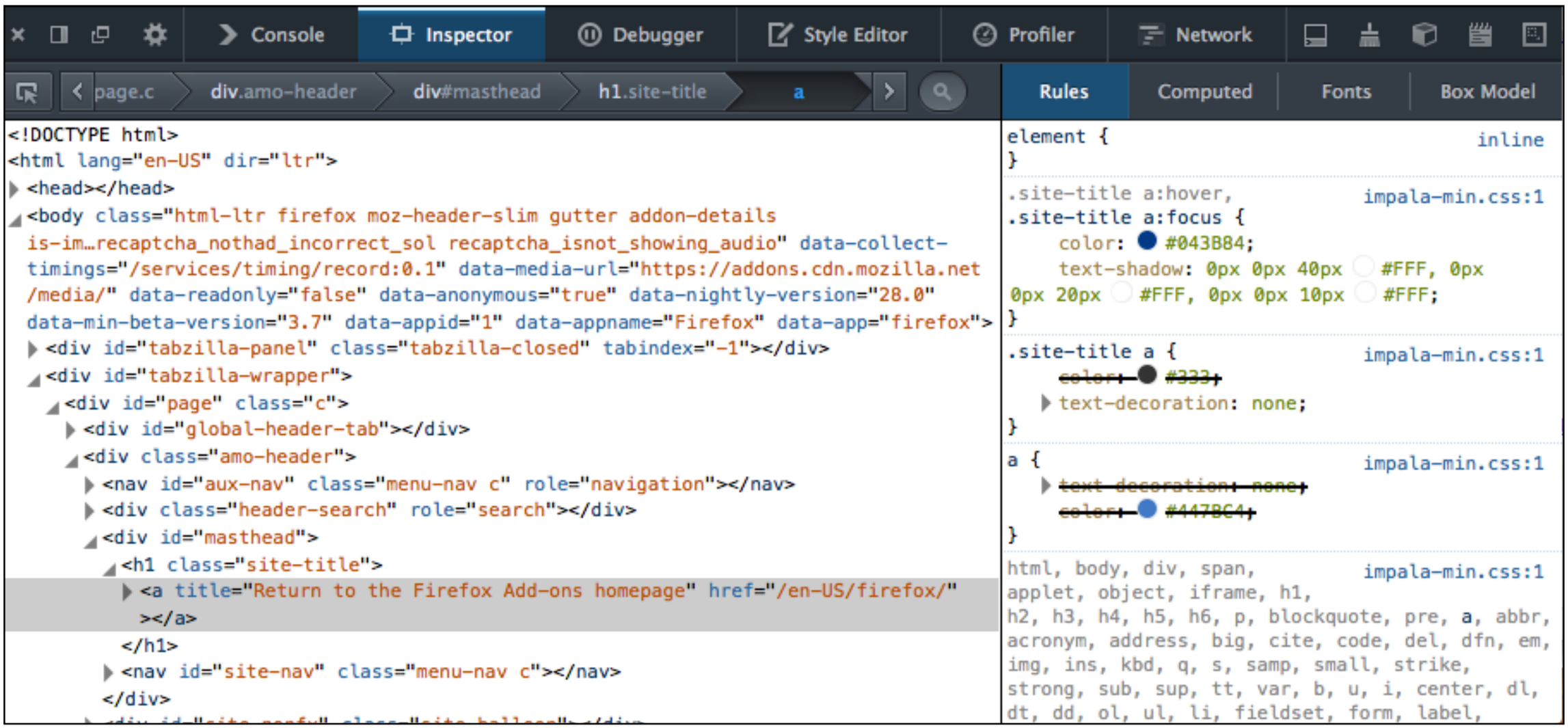


Работа с DOM из консоли

FireFox

-> правой кнопкой мыши по элементу

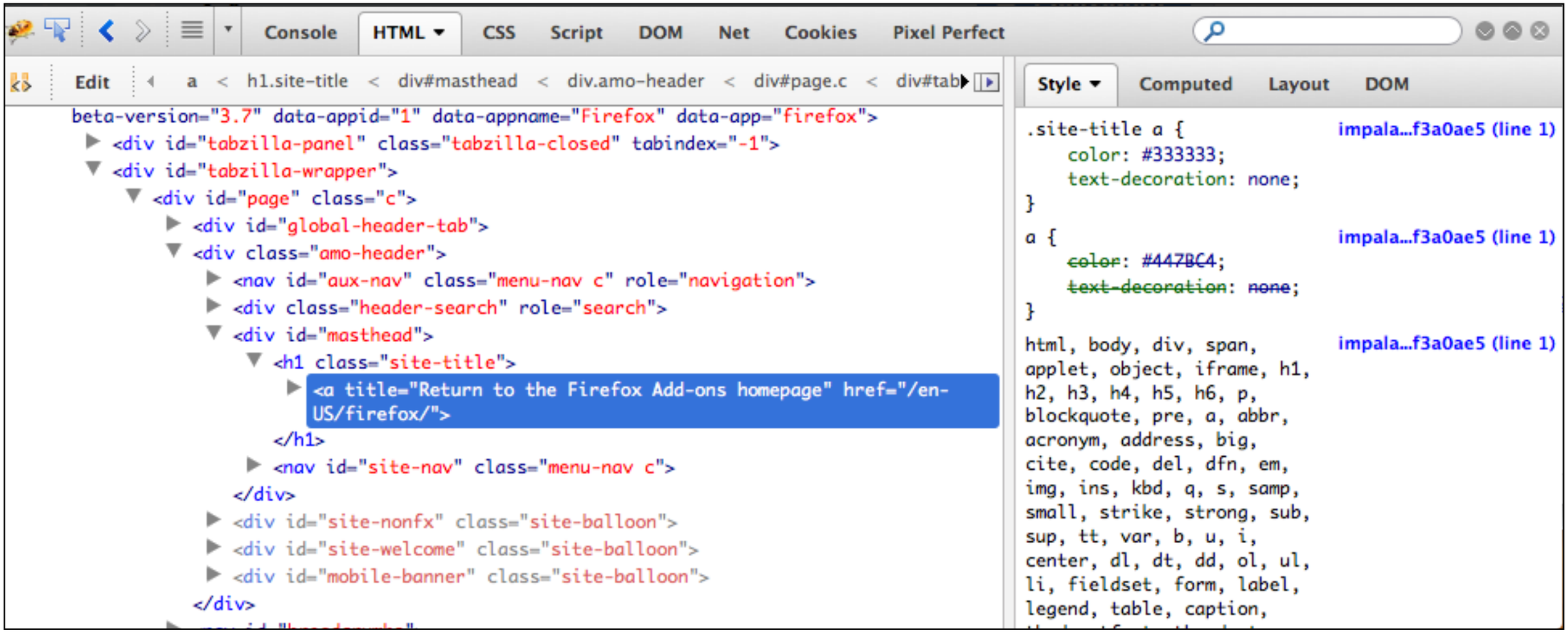
-> Inspect Element => Web Developer Tools



Работа с DOM из консоли

FireFox

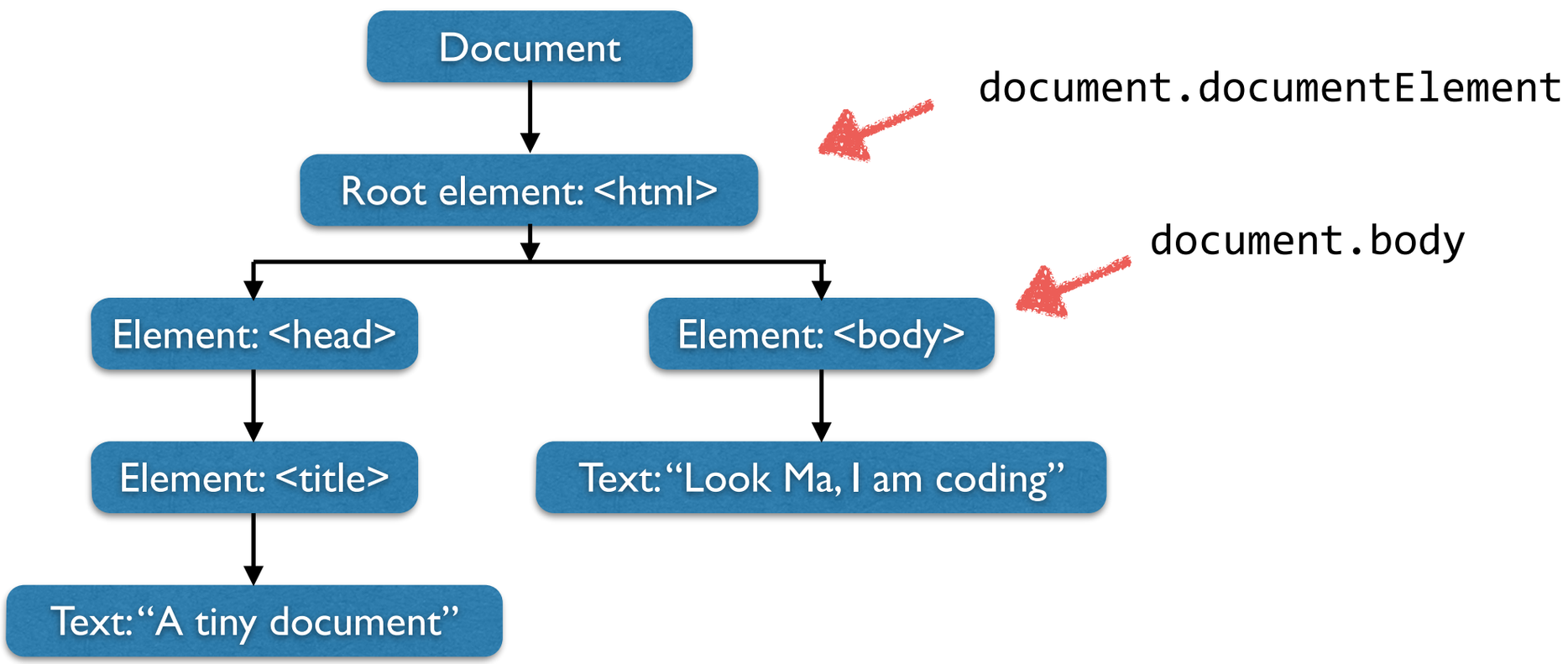
Плагин Firebug: https://addons.mozilla.org/en-US/firefox/addon/firebug/



Работа с DOM

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

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



Работа с 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>
```

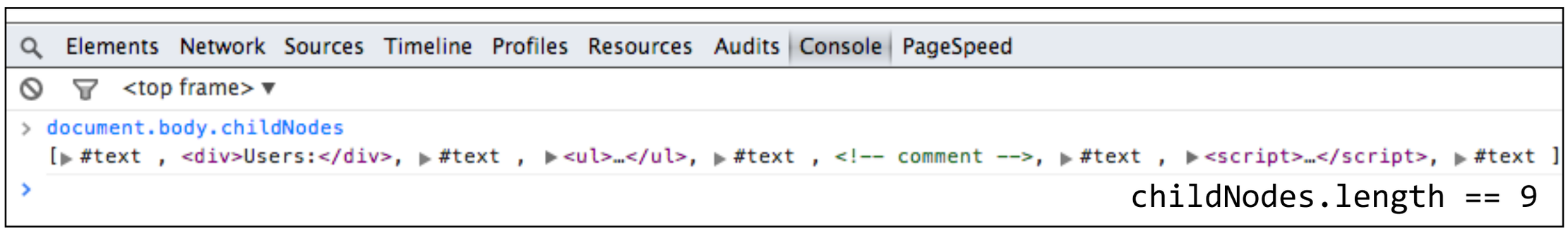
Навигация в DOM

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>
```

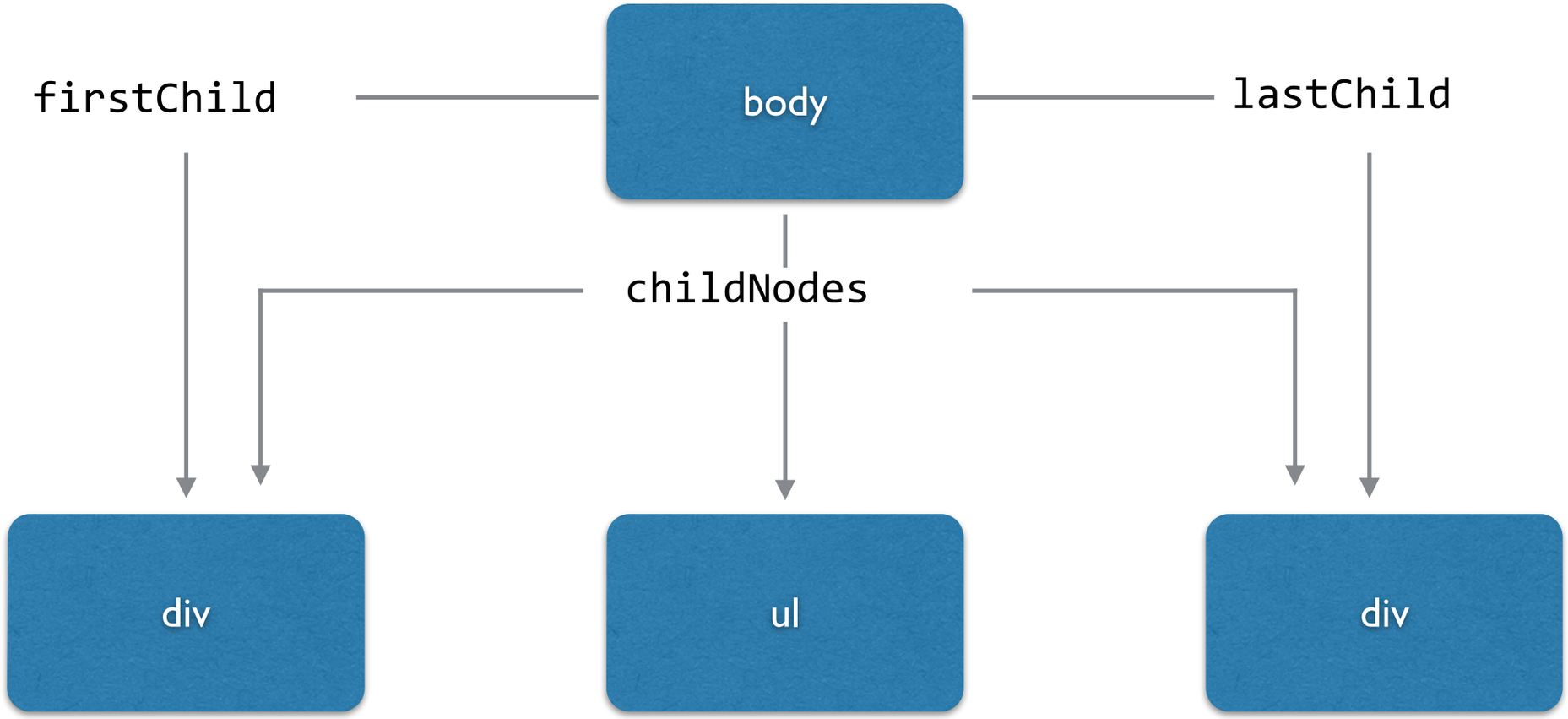
3

?

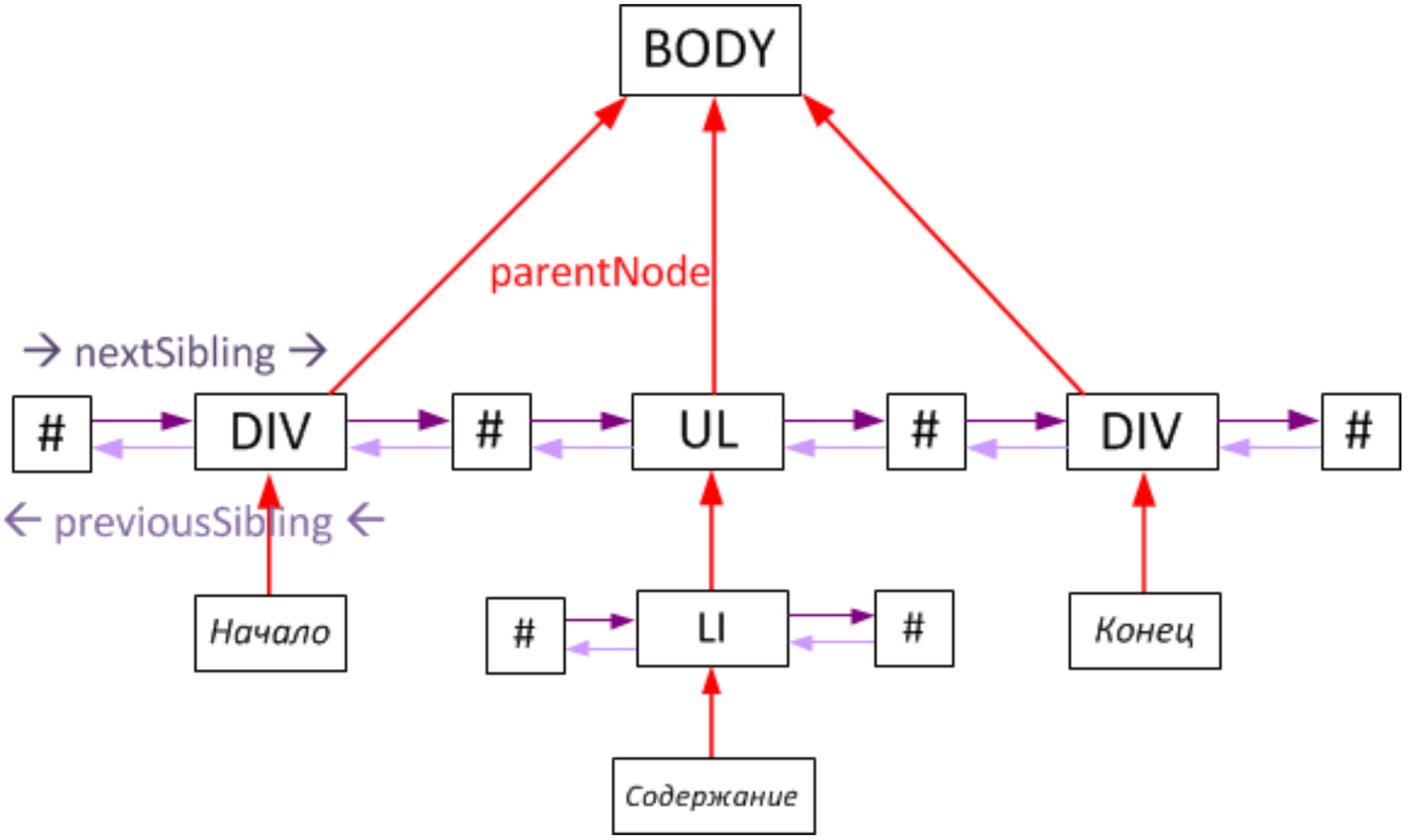
Навигация в DOM

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

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



Навигация в DOM



parentNode – родительский узел
previousSibling и **nextSibling** – левый и правый сосед

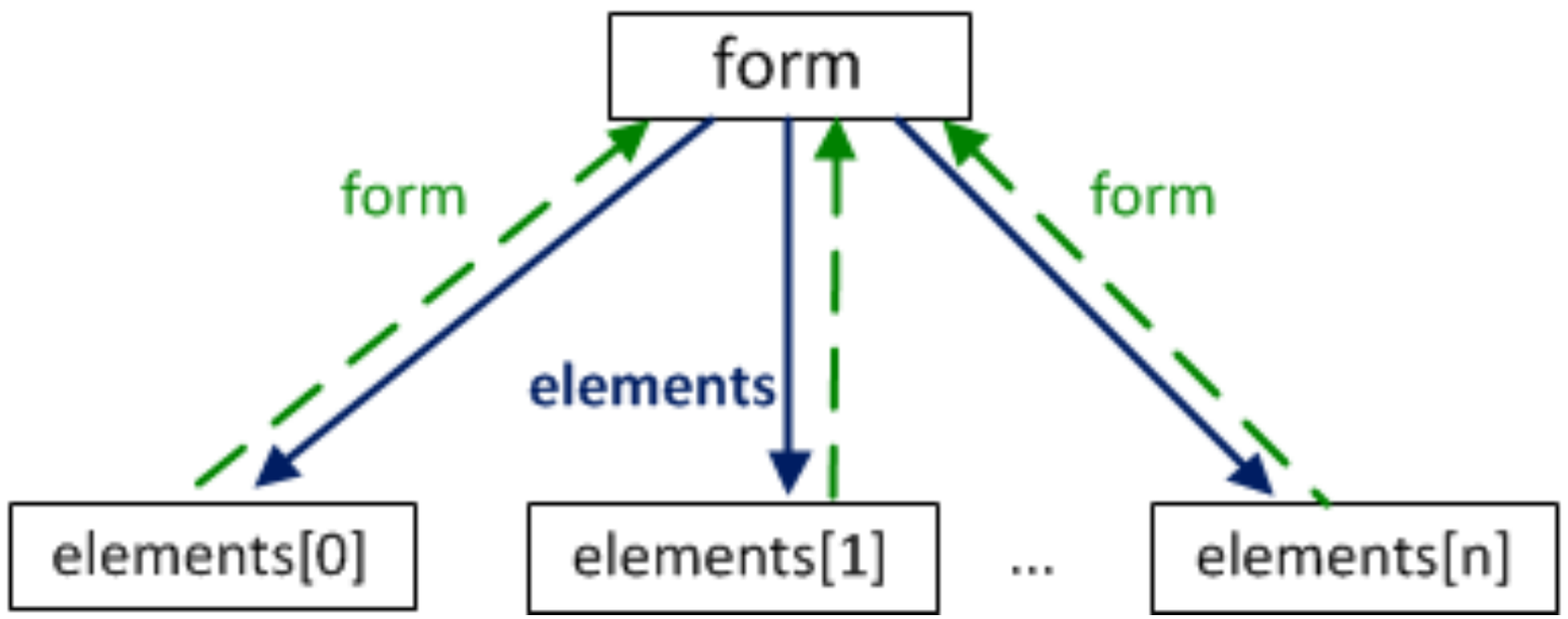
- Все навигационные ссылки – только для чтения
- При изменениях DOM элементов они обновляются автоматически

Формы

`document.forms.my` – форма с именем 'my'
`document.forms[0]` – первая форма в документе

`form.elements` – элементы формы
`form.elements[name]` – коллекция элементов

`element.form` – сама форма



Таблицы

`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

Таблицы

```
<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 HMTL

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

```
<div id=“content”>Some text here</div>
```

```
content = document.getElementById("content");
content.innerHTML = ‘<p>test</p>’;
// Результат:
<div id=“content”><p>test</p></div>
```

Свойства узлов

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

- `id` – идентификатор
- `tagName` – название элемента (“span”)

и другие

Свойства узлов

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 другие

Свойства узлов

Стандартные свойства DOM синхронизируются с атрибутами

```
<a id=“a” href=“#”></a>
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

Задать:

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

name **-** имя атрибута

value **-** значение атрибута

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

Получить значение:

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

name **-** имя атрибута

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

Атрибуты

Проверить наличие:

```
element.hasAttribute(name);
```

name **-** имя атрибута

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

Удалить:

```
element.removeAttribute(name);
```

name **-** имя атрибута

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

removeAttribute(name) вместо `element.setAttribute(name, null);`

При попытке удалить несуществующий атрибут, ошибки не возникнет

Добавление узлов

```
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");
paragraph.appendChild(document.createTextNode("Hi there!"));
content = document.getElementById("content");
document.body.insertBefore(paragraph, content);
```

```
<p>Hi there!</p>
<div id="content">Some text here</div>
```

Добавление узлов

```
child = element.appendChild(child);
```

element – родительский элемент

child – добавляемый элемент типа Node

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

- Помещает последним узлом
- 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

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
```

elements = document.getElementsByClassName(names);

— объект типа HTMLCollection

```
var el1 = document.getElementsByClassName(“red link test”);

var el2 = document.getElementById(“content");
el2.getElementsByClassName(“yellow");
```

Поиск элементов

```
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”),
    cells = table.getElementsByTagName("td");
```

Внешний вид элементов

```
<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


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

d.className;
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

DOM-события, которые инициируются 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

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

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**.

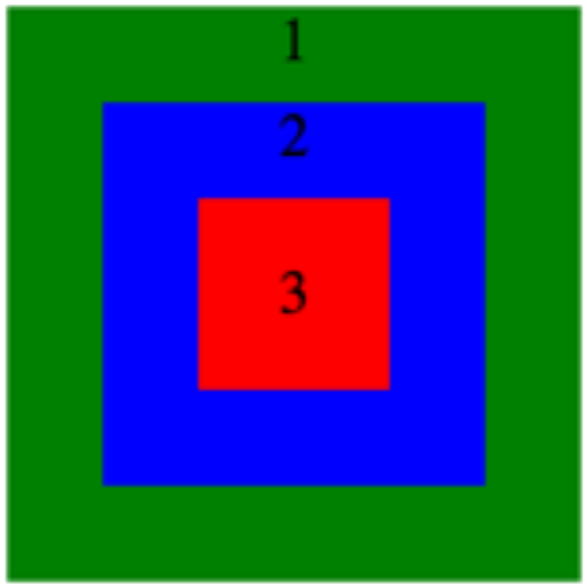
После того, как событие сработает на самом вложенном элементе, оно также сработает на родителях, вверх по цепочке вложенности.

Событийные модели

`event.target` — целевой элемент, самый глубокий, тот который вызывает событие

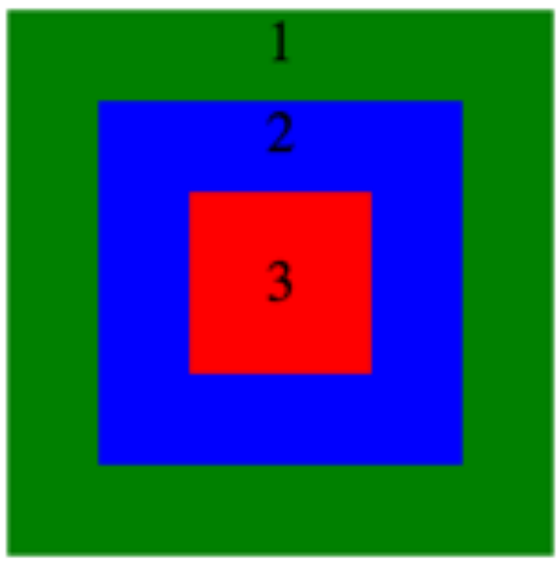
`this` — элемент, на котором сработал обработчик

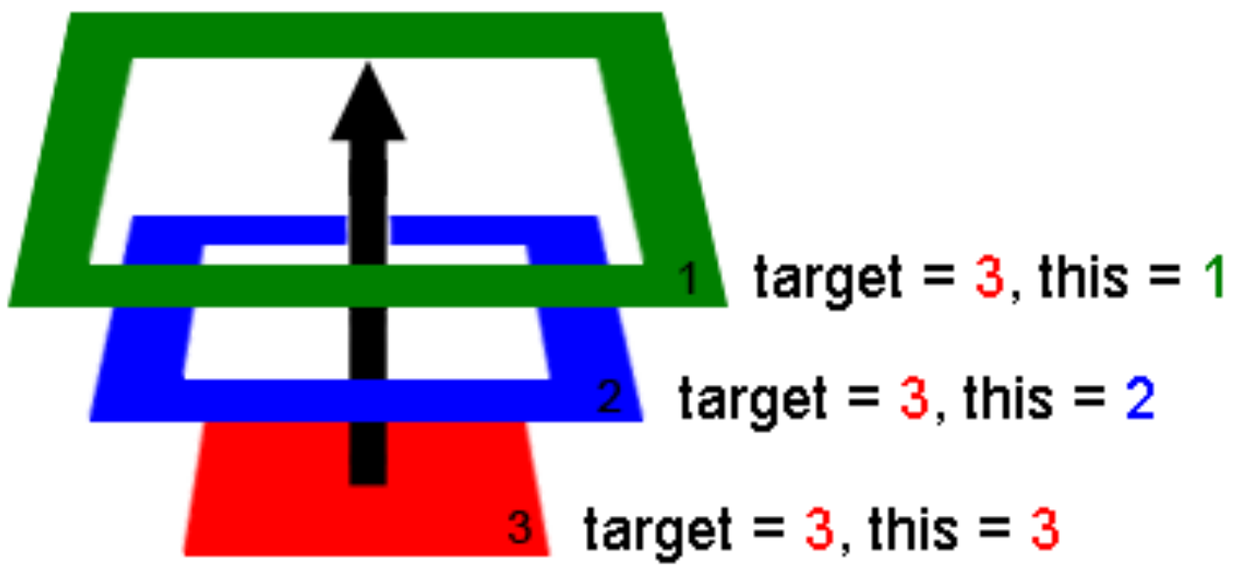
Событийные модели




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

Событийные модели






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

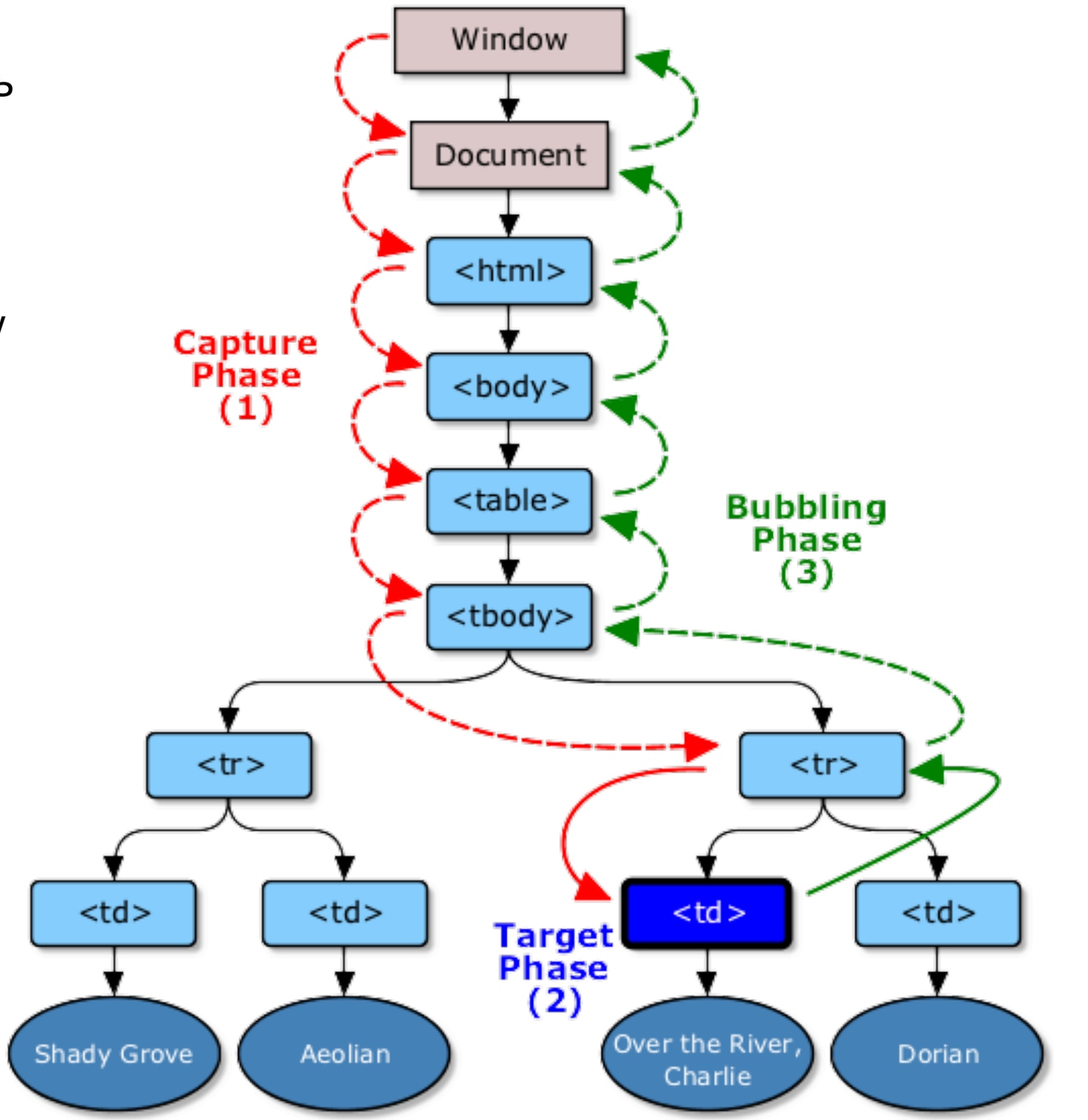
Стадии прохода события

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

1) **«стадия перехвата» (capturing stage)** — событие идет сверху вниз

2) **«стадия цели» (target stage)** — событие достигло целевого элемента

3) **«стадия всплытия» (bubbling stage)** — событие начинает всплывать



Назначение обработчиков событий

- `<a onclick="alert('Clicked!')" href="…"></a>`
- `var elem = document.getElementById('myElement');`
  `elem.onclick = function(event) { … }`

Нельзя назначить больше одного обработчика:

```
<a onclick="alert('Before')" href="…"></a>

var elem = document.getElementsByTagName('a')[0];
elem.onclick = function(event) {
  alert('After');
};
```

addEventListener (кроме IE<9)

Добавляет обработчик:

```
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.

addEventListener (кроме IE<9)

```
<div id="test">
  …
</div>
```

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

removeEventListener (кроме IE<9)

Удаляет ранее добавленный обработчик:

```
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)

removeEventListener (кроме IE<9)

```
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) Обрабатывать события, происходящие на странице или создавать новые
    5) Передвигаться по дереву
3) Существует несколько способ 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®