Курс:
HTML/CSS + JS С НУЛЯ

Тренер:
Илья Литвинов

Лекция 8

WEB ACADEMY
PROGRAMMING COURSES

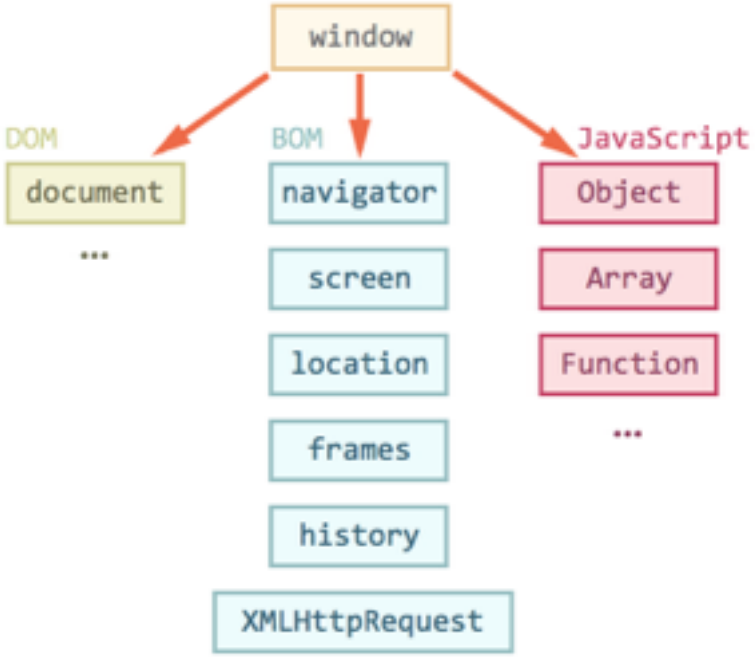
Содержание

- DOM.
- Стили элемента
- Метрики html
- События браузера/работа с событиями браузера

DOM

DOM/BOM

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».



DOM Объектная модель браузера

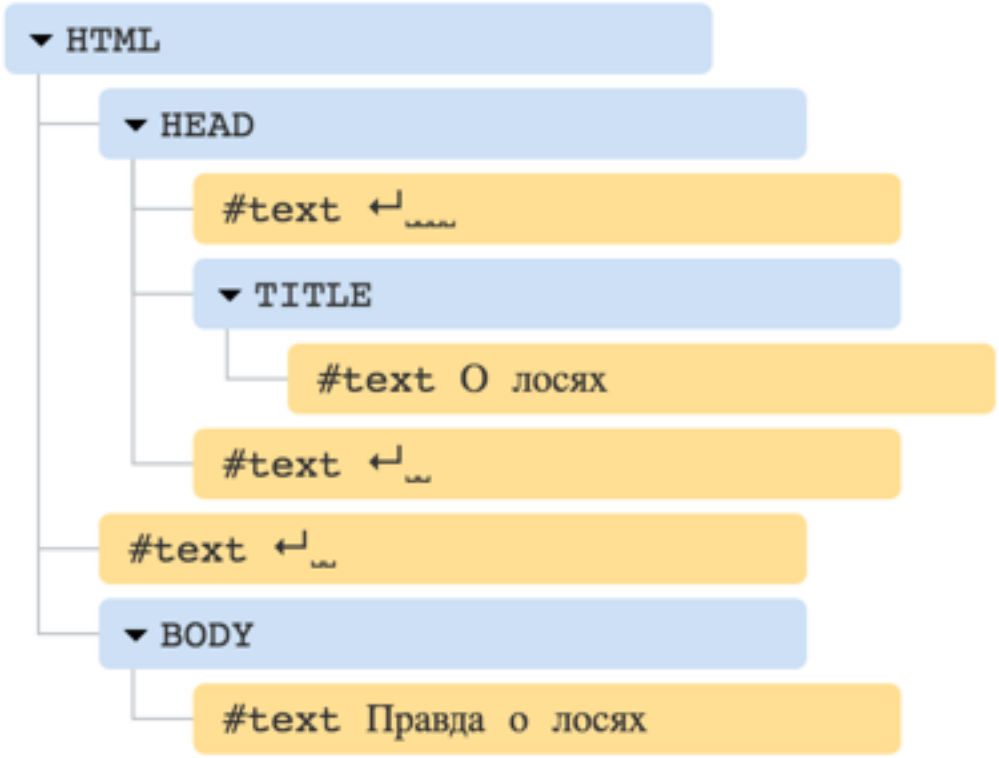
Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов.

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.

DOM Объектная модель браузера

```
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4   <title>О лосях</title>
5 </head>
6 <body>
7   Правда о лосях
8 </body>
9 </html>
```



DOM Объектная модель браузера

В дереве выделено два типа узлов.

- Теги образуют узлы-элементы (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.

- 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.

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

Типы узлов:
Документ – точка входа в DOM.
Элементы – основные строительные блоки.
Текстовые узлы – содержат, собственно, текст.
Комментарии – иногда в них можно включить информацию, которая не будет показана, но доступна из JS.

Возможности 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.

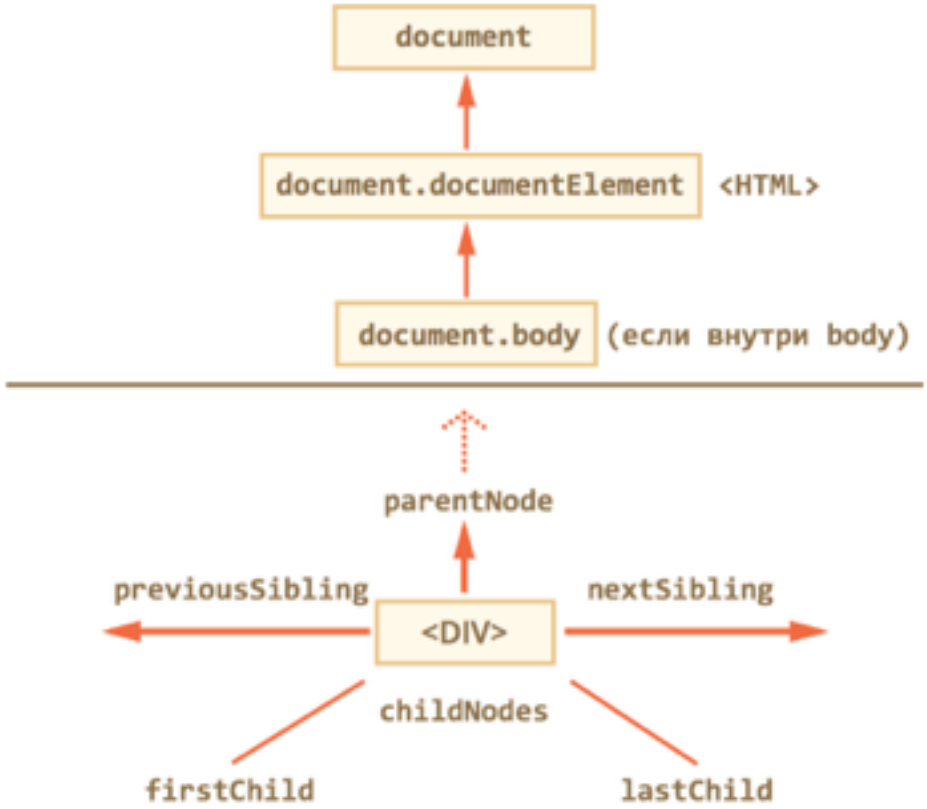
Получив узел, мы можем что-то сделать с ним.

Например, можно поменять цвет BODY и вернуть обратно:

```
1 document.body.style.backgroundColor = 'red';
2 alert( 'Поменяли цвет BODY' );
3
4 document.body.style.backgroundColor = '';
5 alert( 'Сбросили цвет BODY' );
```

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.



Поиск элемента по DOM

- document.getElementById() или просто id
- getElementsByTagName()
- getElementsByName()
- getElementsByClassName()
- querySelectorAll(css)
- querySelector(css)
- elem.closest(css)

Свойство узлов

Тип узла содержится в его свойстве nodeType.
Как правило, мы работаем всего с двумя типами узлов:
- элемент.
- текстовый узел.

```
interface Node {
  // Всевозможные значения nodeType
  const unsigned short ELEMENT_NODE = 1;
  const unsigned short ATTRIBUTE_NODE = 2;
  const unsigned short TEXT_NODE = 3;
  const unsigned short CDATA_SECTION_NODE = 4;
  const unsigned short ENTITY_REFERENCE_NODE = 5;
  const unsigned short ENTITY_NODE = 6;
  const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
  const unsigned short COMMENT_NODE = 8;
  const unsigned short DOCUMENT_NODE = 9;
  const unsigned short DOCUMENT_TYPE_NODE = 10;
  const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
  const unsigned short NOTATION_NODE = 12;
  ...
}
```

Содержимое элементов

- innerHTML
- outerHTML
- textContent
- innerText

Создание элемента

document.createElement(tag)

Создает новый элемент с указанным тегом:

```
1 var div = document.createElement('div');
```

document.createTextNode(text)

Создает новый *текстовый* узел с данным текстом:

```
1 var textElem = document.createTextNode('Тут был я');
```

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

Добавление элемента в DOM

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.

parentElem.appendChild(elem)

```
<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>
```

Добавление элемента в DOM

parentElem.insertBefore(elem, nextSibling)

```
<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>
```

Клонирование узлов

node.cloneNode(deep)

```
11 <body>
12   <h3>Моя страница</h3>
13 </body>
14
15 <script>
16   var div = document.createElement('div');
17   div.className = "alert alert-success";
18   div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";
19
20   document.body.insertBefore(div, document.body.firstChild);
21
22   // создать копию узла
23   var div2 = div.cloneNode(true);
24   // копию можно подправить
25   div2.querySelector('strong').innerHTML = 'Супер!';
26   // вставим её после текущего сообщения
27   div.parentNode.insertBefore(div2, div.nextSibling);
28 </script>
```

Удаление узлов

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.

```
<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>
```

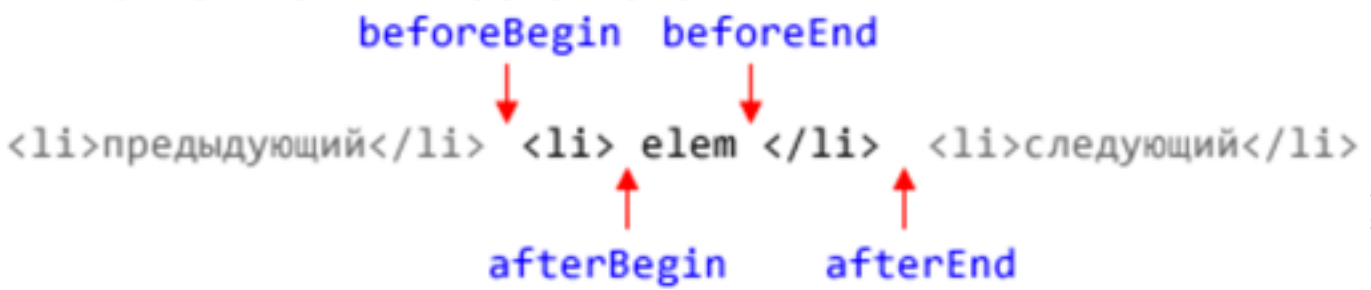
Мультивставка

elem.insertAdjacentHTML(where, html);
позволяет вставлять произвольный HTML в любое место документа, в том числе и между узлами!

html
Строка HTML, которую нужно вставить

where :Куда по отношению к elem вставлять строку. Всего четыре варианта:

1. `beforeBegin` -- перед `elem`.
2. `afterBegin` -- внутрь `elem`, в самое начало.
3. `beforeEnd` -- внутрь `elem`, в конец.
4. `afterEnd` -- после `elem`.



Мультивставка

document.createDocumentFragment();

Вставить пачку узлов единовременно поможет DocumentFragment. Это особенный кросс-браузерный DOM-объект, который похож на обычный DOM-узел, но им не является.

```
1 fragment.appendChild(node);
```

У DocumentFragment нет обычных свойств DOM-узлов, таких как innerHTML, tagName и т.п. Это не узел.

Его «Фишка» заключается в том, что когда DocumentFragment вставляется в DOM – то он исчезает, а вместо него вставляются его дети. Это свойство является уникальной особенностью DocumentFragment.

Работа со стилям

Стили

element.style возвращает объект, который дает доступ к стилю элемента на чтение и запись.
С его помощью можно изменять большинство CSS-свойств, например element.style.width="100px" работает так, как будто у элемента в атрибуте прописано style="width:100px".

Для свойств, названия которых состоят из нескольких слов, используется верблюжья аннотация вотТакаяЗапись:

```
1 background-color  => elem.style.backgroundColor
2 z-index           => elem.style.zIndex
3 border-left-width => elem.style.borderLeftWidth
```

Чтобы сбросить поставленный стиль, присваивают в style пустую строку: elem.style.width=""

Полный стили getComputedStyle()

Для того, чтобы получить текущее используемое значение свойства, используется метод window.getComputedStyle

```
1 getComputedStyle(element[, pseudo])
```

element
Элемент, значения для которого нужно получить
pseudo
Указывается, если нужен стиль псевдо-элемента, например "::before". Пустая строка или отсутствие аргумента означают сам элемент.

Работа с элементами DOM

С генерированный стиль эл-та

Свойство style дает доступ только к той информации, которая хранится в elem.style.

Он не скажет ничего об отступе, если он появился в результате наложения CSS или встроенных стилей браузера:
А если мы хотим, например, сделать анимацию и плавно увеличивать marginTop от текущего значения? Как нам сделать это? Ведь для начала нам надо это текущее значение получить.

```
1 getComputedStyle(element[, pseudo])
```

element
Элемент, значения для которого нужно получить

pseudo
Указывается, если нужен стиль псевдо-элемента, например "::before". Пустая строка или отсутствие аргумента означают сам элемент.

getComputedStyle

```
<style>
  body {
    margin: 10px
  }
</style>

<body>

  <script>
    var computedStyle = getComputedStyle(document.body);
    alert( computedStyle.marginTop ); // выведет отступ в пикселях
    alert( computedStyle.color ); // выведет цвет
  </script>

</body>
```

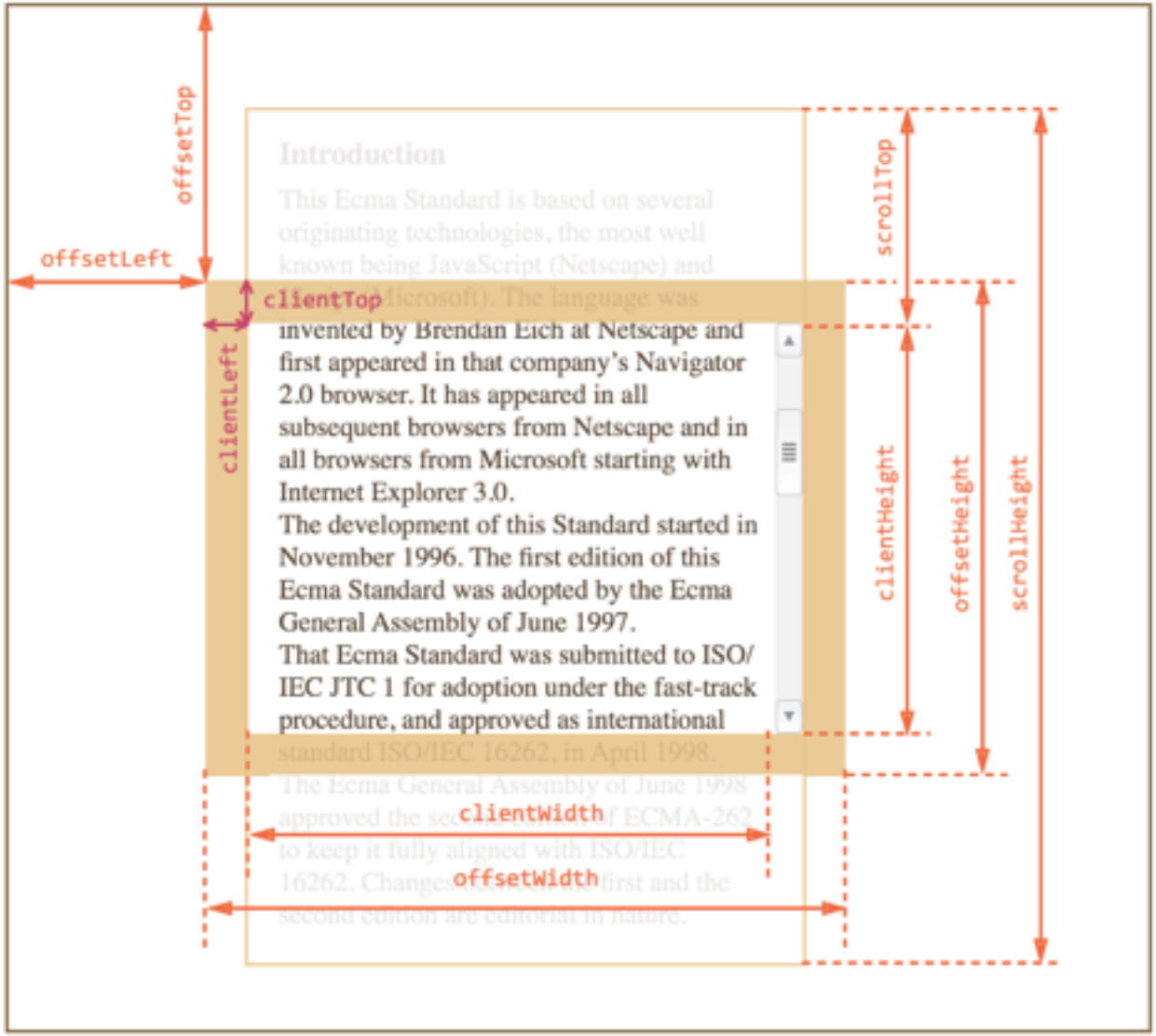
HTML метрики

HTML метрики

У элементов существует ряд свойств, содержащих их внешние и внутренние размеры. Мы будем называть их «метриками».

Метрики, в отличие от свойств CSS, содержат числа, всегда в пикселях и без единиц измерения на конце.

HTML метрики

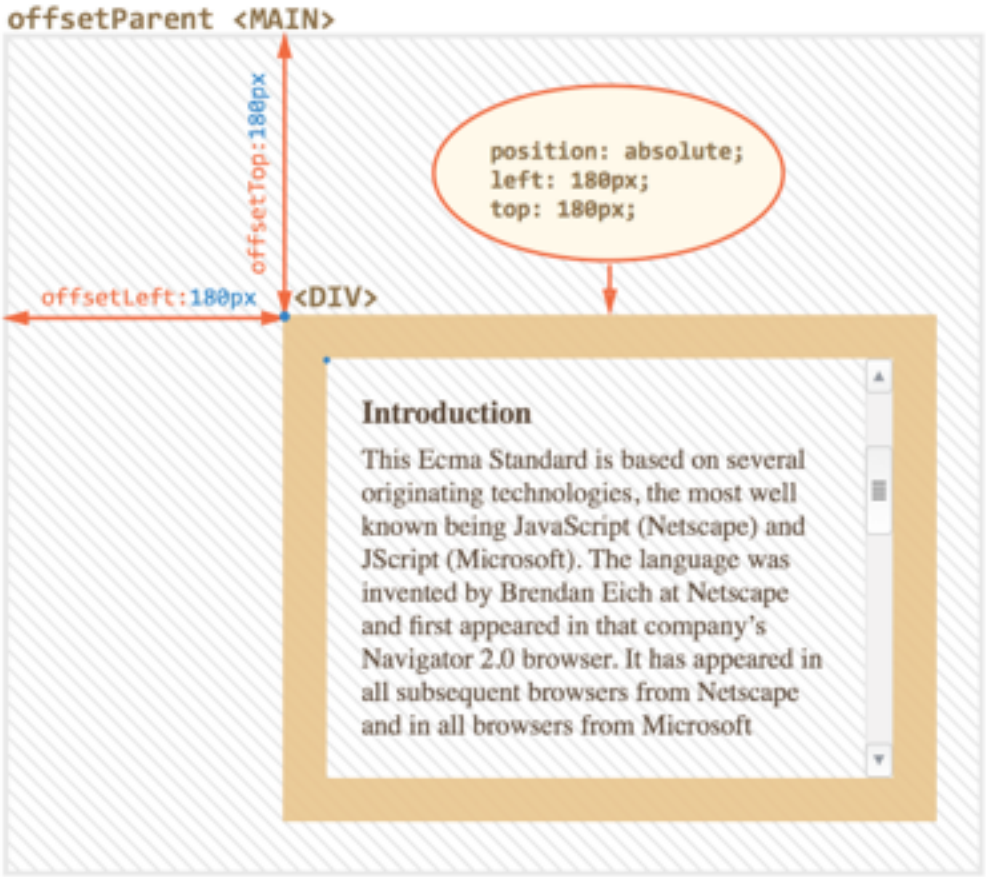


offsetParent/offsetLeft/Top

```
<main style="position: relative">
  <form>
    <div id="example" style="position: absolute; left: 180px; top: 180px">...</div>
  </form>
</main>
```

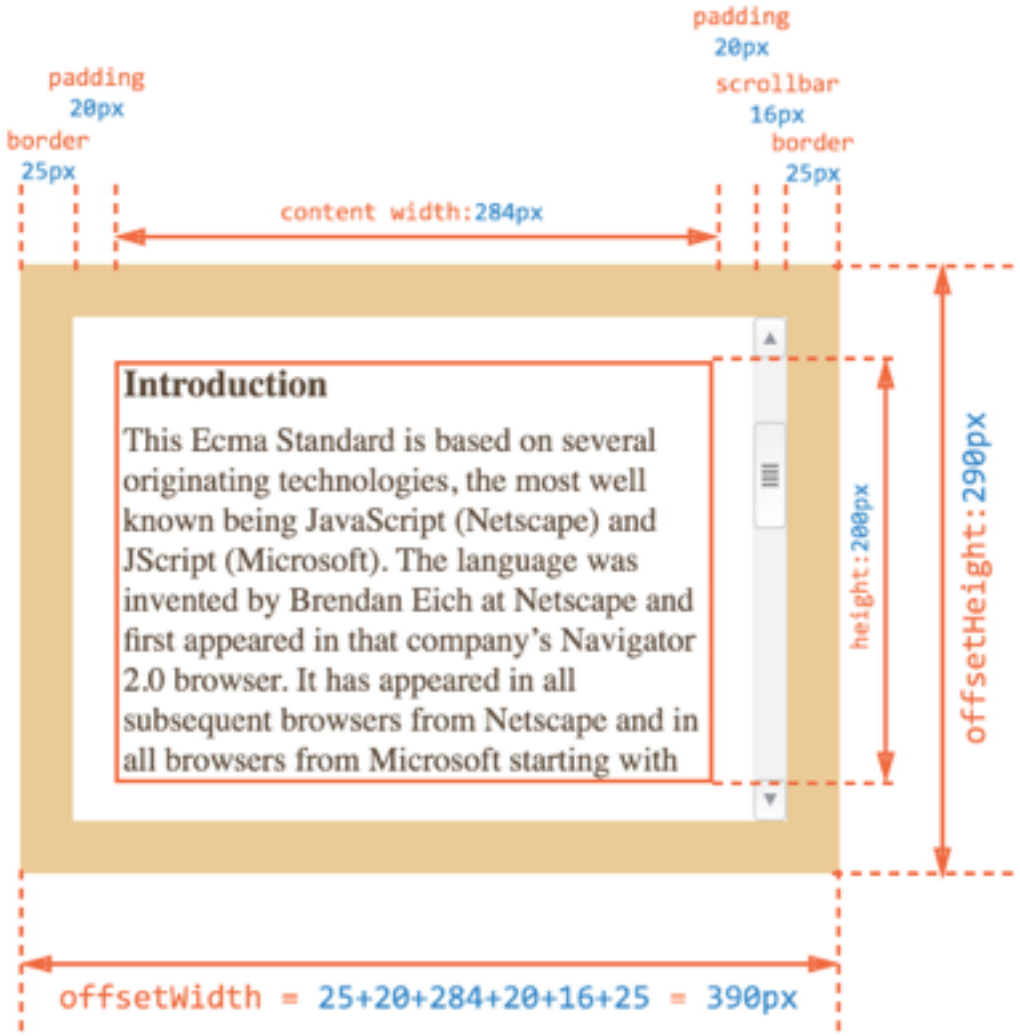
В offsetParent находится ссылка на родительский элемент в смысле отображения на странице.

Свойства offsetLeft/Top задают смещение относительно offsetParent.



offsetWidth/Height

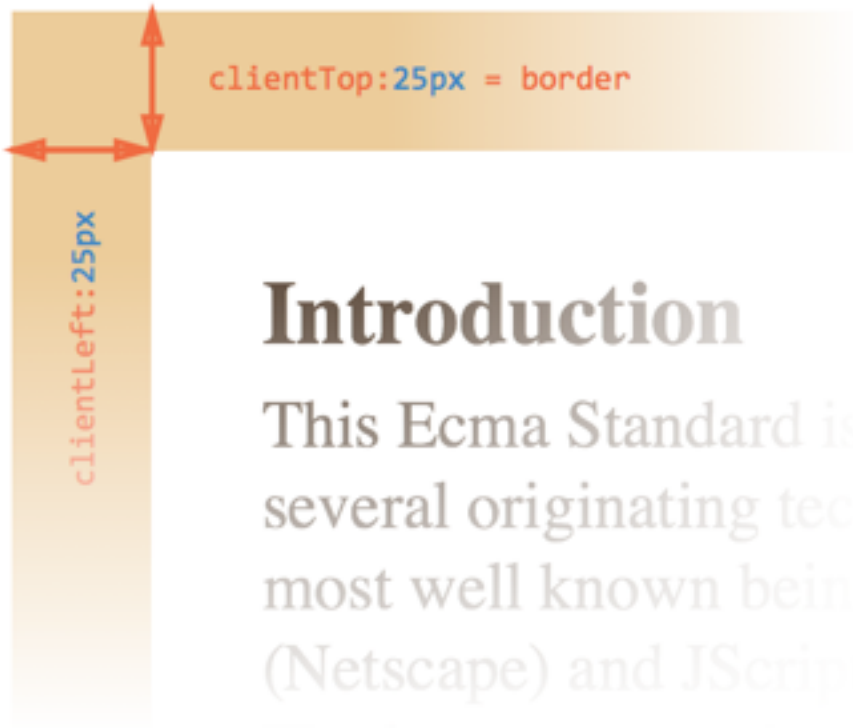
Содержат «внешнюю» ширину/высоту элемента, то есть его полный размер, включая рамки border.



clientTop/Left

Внутри элемента у нас рамки border.

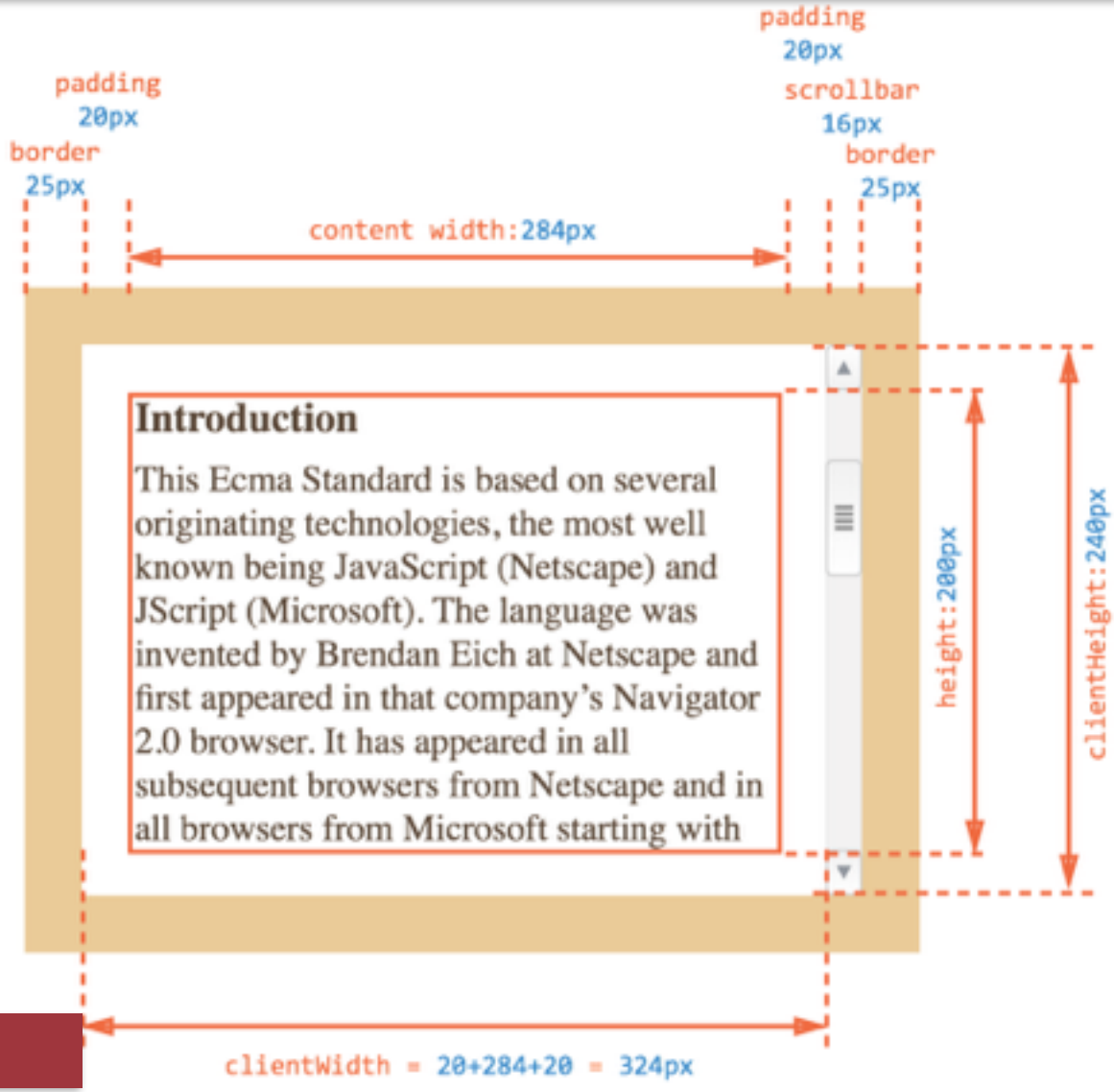
Для них есть свойства-метрики clientTop и clientLeft.



clientWidth/Height

Эти свойства – размер элемента внутри рамок border.

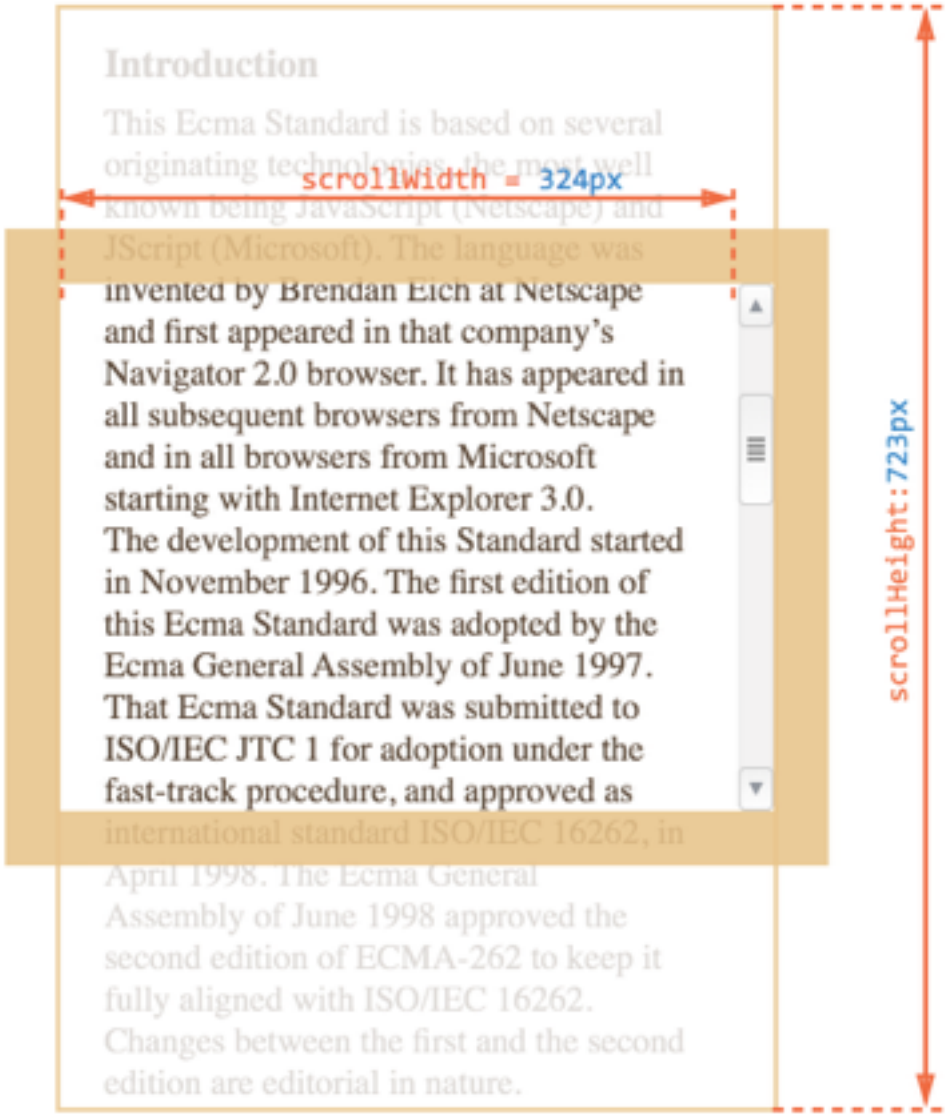
Они включают в себя ширину содержимого width вместе с полями padding, но без прокрутки.



scrollWidth/Height

Эти свойства – аналоги clientWidth/clientHeight, но с учетом прокрутки.

Свойства clientWidth/clientHeight относятся только к видимой области элемента, а scrollWidth/scrollHeight добавляют к ней прокрученную (которую не видно) по горизонтали/вертикали



События браузера

События браузера

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. Посмотрим список самых часто используемых, пока просто для ознакомления»

Типы событий

События мыши:

- click – происходит, когда кликнули на элемент левой кнопкой мыши
- contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
- mouseover – возникает, когда на элемент наводится мышь
- mousedown и mouseup – когда кнопку мыши нажали или отжали
- mousemove – при движении мыши

Клавиатурные события:

- keydown – когда посетитель нажимает клавишу
- keyup – когда посетитель отпускает клавишу

Типы событий

События на элементах управления:

submit – посетитель отправил форму <form>
focus – посетитель фокусируется на элементе, например нажимает на <input>

События CSS:

transitionend – когда CSS-анимация завершена.

Назначение обработчиков событий

Событию можно назначить обработчик, то есть функцию, которая сработает, как только событие произошло.
Именно благодаря обработчикам JavaScript-код может реагировать на действия посетителя. Существует несколько способов назначить обработчик на событие.

- Атрибут HTML
- Использование св-ва DOM обьекта el.onclick()
- el.addEventListener(event, handler, [phase])/removeEventListener()

Атрибут HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.
Например, чтобы прикрепить click-событие к input кнопке, можно присвоить обработчик onclick, вот так:

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">
```

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="utf-8">

  <script>
    function countRabbits() {
      for(var i=1; i<=3; i++) {
        alert("Кролик номер " + i);
      }
    }
  </script>
</head>
<body>
  <input type="button" onclick="countRabbits()" value="Считать кроликов!"/>
</body>
</html>
```

Использование св-ва DOM обьекта

Можно назначать обработчик, используя свойство DOM-элемента on<событие>

```
<input id="elem" type="button" value="Нажми меня" />
<script>
  elem.onclick = function() {
    alert( 'Спасибо' );
  };
</script>
```

Так как DOM-свойство onclick, в итоге, одно, то назначить более одного обработчика так нельзя.

```
<input type="button" id="elem" onclick="alert('До')" value="Нажми меня" />
<script>
  elem.onclick = function() { // перезапишет существующий обработчик
    alert( 'После' ); // выведется только это
  };
</script>
```

Использование св-ва DOM обьекта

Внутри обработчика события this ссылается на текущий элемент, то есть на тот, на котором он сработал.

Это можно использовать, чтобы получить свойства или изменить элемент.

В коде ниже button выводит свое содержимое, используя this.innerHTML:

```
1 <button onclick="alert(this.innerHTML)">Нажми меня</button>
```

el.addEventListener()

Методы addEventListener и removeEventListener являются современным способом назначить или удалить обработчик, и при этом позволяют использовать сколько угодно любых обработчиков.

```
1 element.addEventListener(event, handler[, phase]);
```

event
Имя события, например click
handler
Ссылка на функцию, которую надо поставить обработчиком.
phase
Необязательный аргумент, «фаза», на которой обработчик должен сработать

```
1 // передать те же аргументы, что были у addEventListener
2 element.removeEventListener(event, handler[, phase]);
```

el.addEventListener()

Метод addEventListener позволяет добавлять несколько обработчиков на одно событие одного элемента

```
<input id="elem" type="button" value="Нажми меня"/>

<script>
  function handler1() {
    alert('Спасибо!');
  };

  function handler2() {
    alert('Спасибо ещё раз!');
  }

  elem.onclick = function() { alert("Привет"); };
  elem.addEventListener("click", handler1); // Спасибо!
  elem.addEventListener("click", handler2); // Спасибо ещё раз!
</script>
```

Конец