# JavaScript - координаты

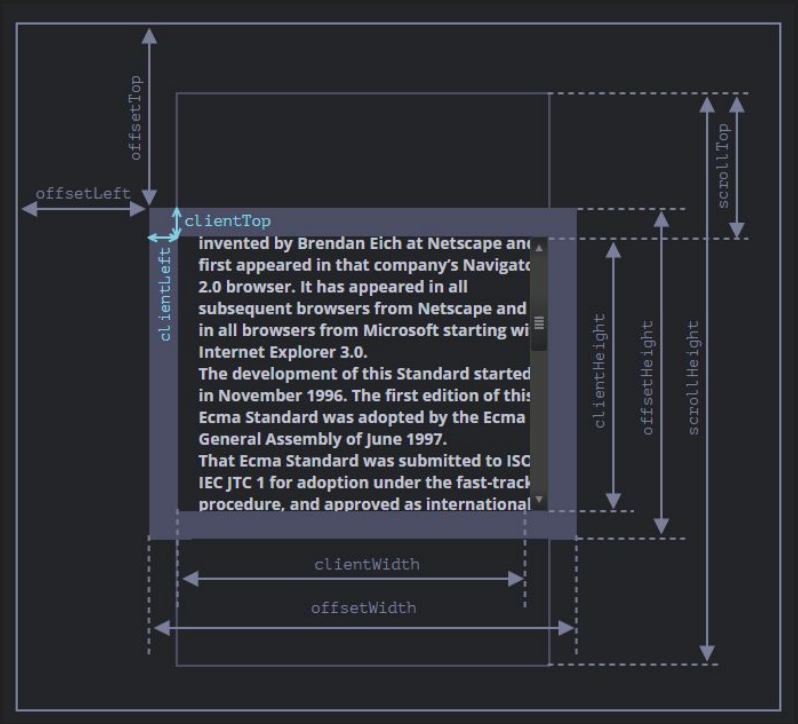
размеры и прокрутка элементов,
размеры и прокрутка окна,
координаты, canvas

# Размеры и прокрутка элементов

Существует множество JavaScript-свойств, которые позволяют считывать информацию об элементе: ширину, высоту и другие геометрические характеристики.

Они часто требуются, когда нам нужно передвигать или позиционировать элементы с помощью JavaScript.

- Мерики
- offsetParent, offsetLeft/Top
- offsetWidth/Height
- clientTop/Left
- clientWidth/Height
- scrollWidth/Height
- scrollLeft/scrollTop



# offsetParent, offsetLeft/Top

В свойстве offsetParent находится предок элемента, который используется внутри браузера для вычисления координат при рендеринге.

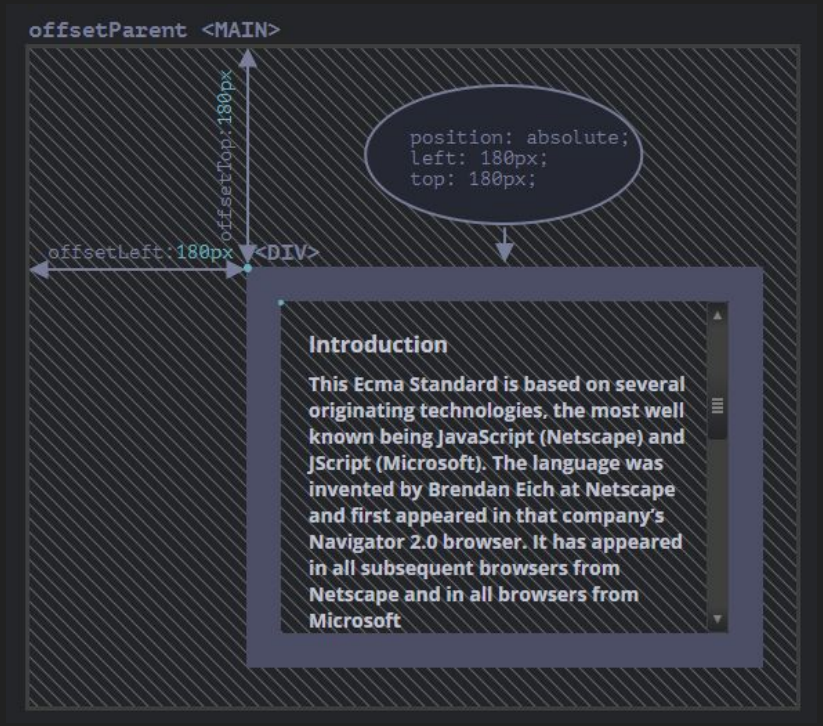
**То есть, ближайший предок, который удовлетворяет следующим условиям:**

- Является CSS-позиционированным (CSS-свойство position равно absolute, relative, fixed или sticky),
- или <td>, <th>, <table>,
- или <body>.

Свойства offsetLeft/offsetTop содержат координаты x/y относительно верхнего левого угла offsetParent.

**Существует несколько ситуаций, когда offsetParent равно null:**

- Для скрытых элементов (с CSS-свойством display:none или когда его нет в документе).
- Для элементов <body> и <html>.
- Для элементов с position:fixed.



# offsetWidth/Height

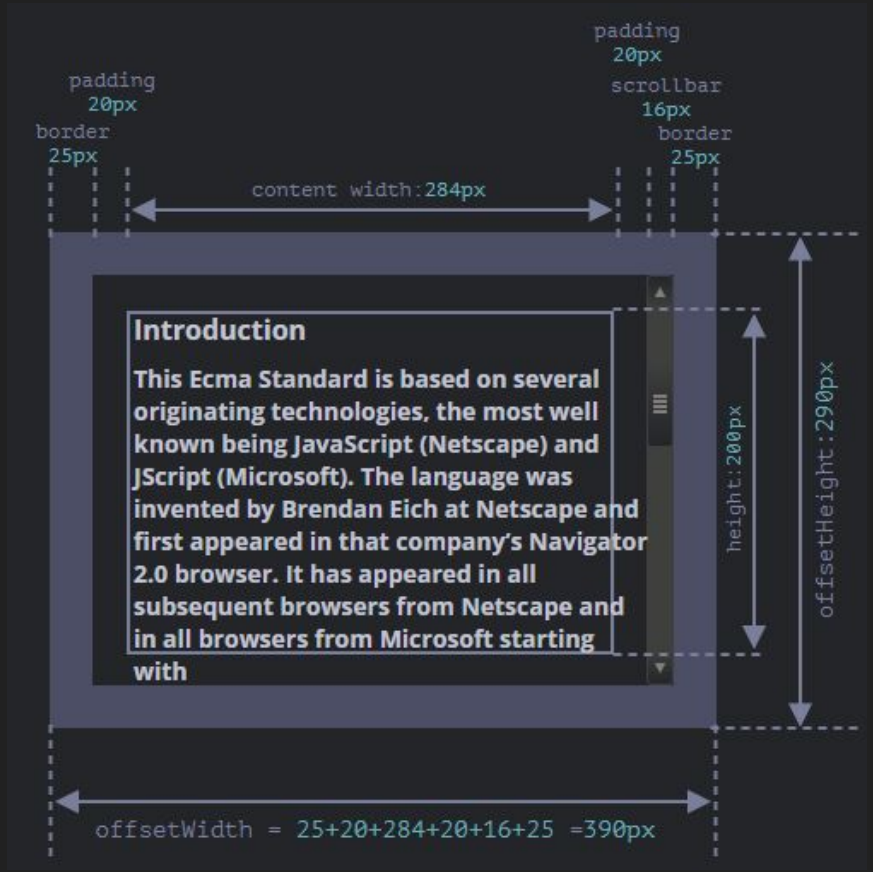
Эти два свойства – самые простые. Они содержат «внешнюю» ширину/высоту элемента, то есть его полный размер, включая рамки.

**Метрики для не показываемых элементов равны нулю.**

Координаты и размеры в JavaScript устанавливаются только для видимых элементов.

Если элемент (или любой его родитель) имеет display:none или отсутствует в документе, то все его метрики равны нулю (или null, если это offsetParent).

Например, свойство offsetParent равно null, а offsetWidth и offsetHeight равны 0, когда мы создали элемент, но ещё не вставили его в документ, или если у элемента (или у его родителя) display:none.



# clientTop/Left

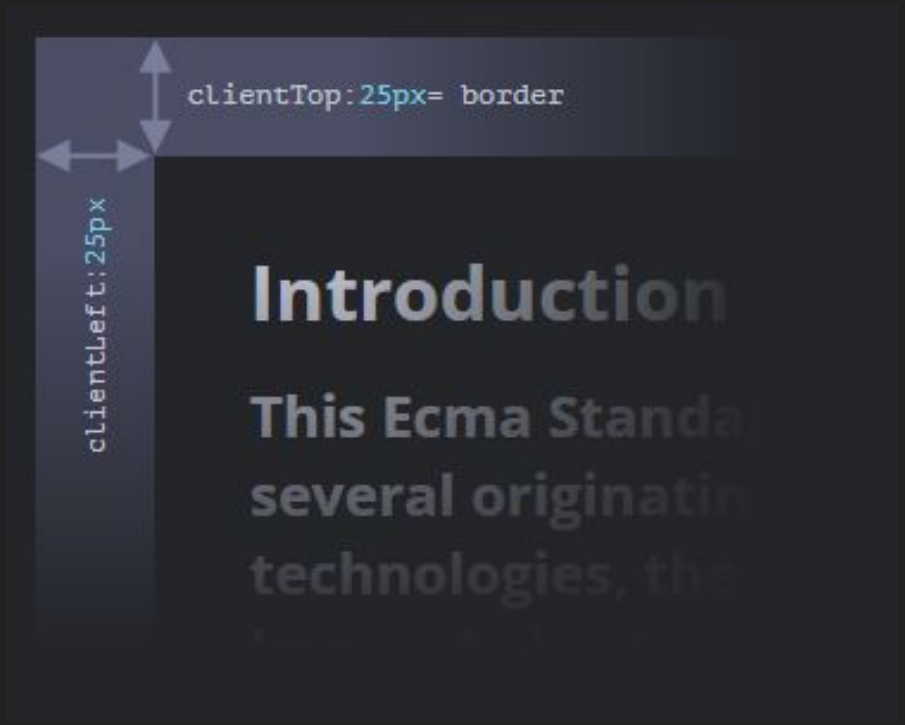
Внутри элемента у нас рамки (border).

Для них есть свойства-метрики clientTop и clientLeft.

В нашем примере:

- clientLeft = 25 – ширина левой рамки
- clientTop = 25 – ширина верхней рамки

Но на самом деле эти свойства – вовсе не ширины рамок, а отступы внутренней части элемента от внешней.



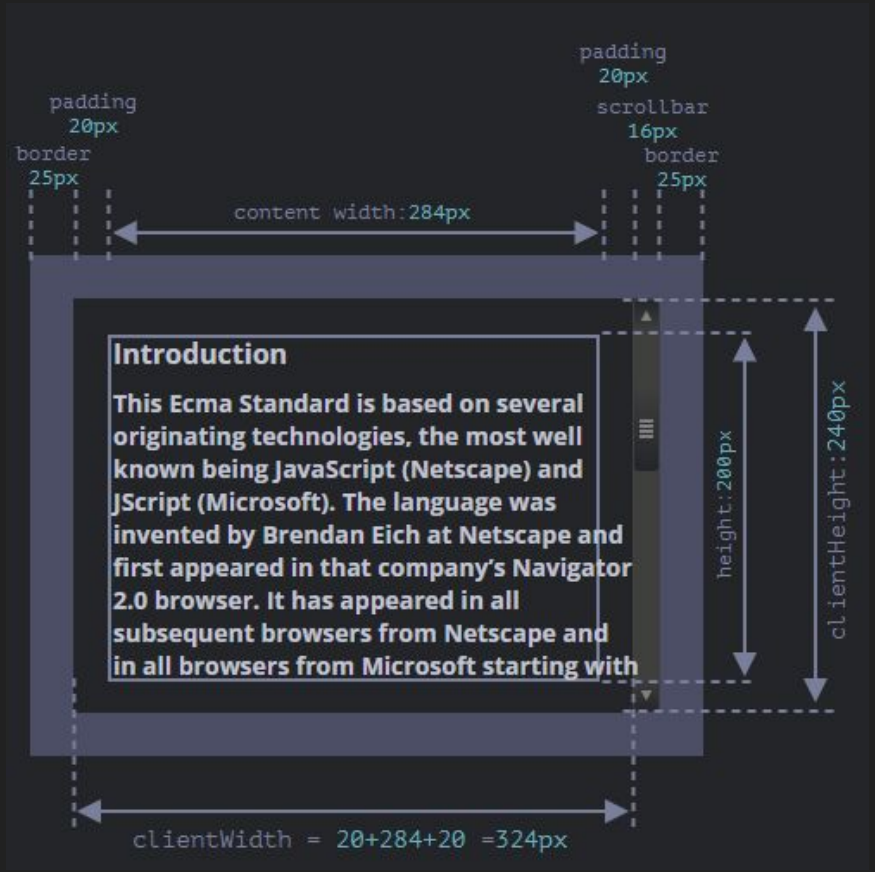
# clientWidth/Height

Эти свойства – размер области внутри рамок элемента.

Они включают в себя ширину области содержимого вместе с внутренними отступами padding, но без прокрутки:

Если нет внутренних отступов padding, то clientWidth/Height в точности равны размеру области содержимого внутри рамок за вычетом полосы прокрутки (если она есть).

Поэтому в тех случаях, когда мы точно знаем, что отступов нет, можно использовать clientWidth/clientHeight для получения размеров внутренней области содержимого.

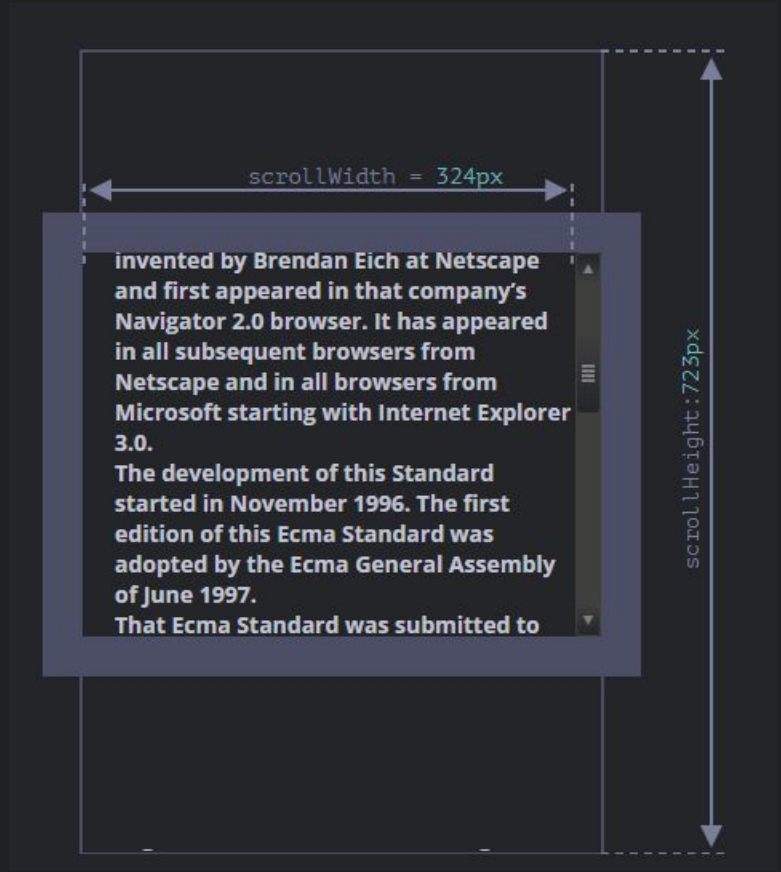


# scrollWidth/Height

Эти свойства – как clientWidth/clientHeight, но также включают в себя прокрученную (которую не видно) часть элемента.

Пример:

- scrollHeight = 723 – полная внутренняя высота, включая прокрученную область.
- scrollWidth = 324 – полная внутренняя ширина, в данном случае прокрутки нет, поэтому она равна clientWidth.



# scrollLeft/scrollTop

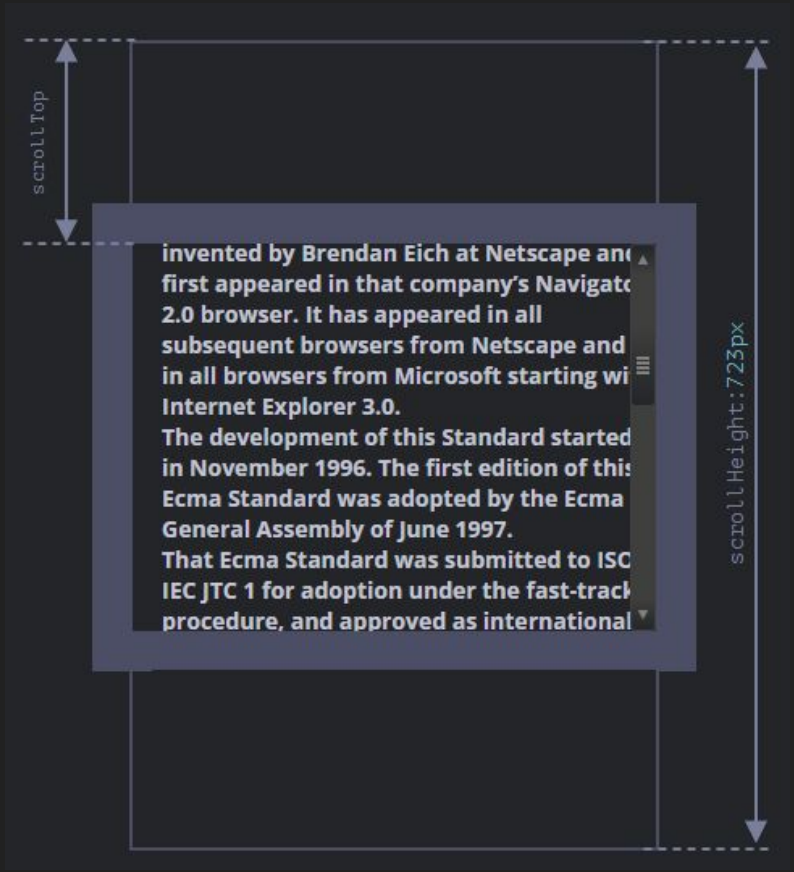
Свойства scrollLeft/scrollTop – ширина/высота невидимой, прокрученной в данный момент, части элемента слева и сверху.

Другими словами, свойство scrollTop – это «сколько уже прокручено вверх».

Свойства scrollLeft/scrollTop можно изменять

В отличие от большинства свойств, которые доступны только для чтения, значения scrollLeft/scrollTop можно изменять, и браузер выполнит прокрутку элемента.

Установка значения scrollTop на 0 или на большое значение, такое как 1e9, прокрутит элемент в самый верх/низ соответственно.



# Не стоит брать width/height из CSS

Почему мы должны использовать свойства-метрики вместо этого? На то есть две причины:

1. Во-первых, CSS-свойства width/height зависят от другого свойства – box-sizing, которое определяет, «что такое», собственно, эти CSS-ширина и высота. Получается, что изменение box-sizing, к примеру, для более удобной вёрстки, сломает такой JavaScript.
2. Во-вторых, CSS свойства width/height могут быть равны auto, например, для инлайнового элемента. Конечно, с точки зрения CSS width:auto – совершенно нормально, но нам-то в JavaScript нужен конкретный размер в px, который мы могли бы использовать для вычислений. Получается, что в данном случае ширина из CSS вообще бесполезна.

# Размеры и прокрутка окна

Как узнать ширину и высоту окна браузера? Как получить полную ширину и высоту документа, включая прокрученную часть? Как прокрутить страницу с помощью JavaScript?

Для большинства таких запросов мы можем использовать корневой элемент документа document.documentElement, который соответствует тегу <html>. Однако есть дополнительные методы и особенности, которые необходимо учитывать.

- Ширина/высота окна
- Ширина/высота документа
- Получение текущей прокрутки
- Прокрутка: scrollTo, scrollBy
- scrollIntoView
- Запретить прокрутку

# Ширина/высота окна

Чтобы получить ширину/высоту окна, можно взять свойства clientWidth/clientHeight из document.documentElement:

Браузеры также поддерживают свойства window.innerWidth/innerHeight. Вроде бы, похоже на то, что нам нужно. Почему же не использовать их?

Если есть полоса прокрутки, и она занимает какое-то место, то свойства clientWidth/clientHeight указывают на ширину/высоту документа без нее (за ее вычетом). Иными словами, они возвращают высоту/ширину видимой части документа, доступной для содержимого.

А window.innerWidth/innerHeight включают в себя полосу прокрутки.


**DOCTYPE – это важно**
Обратите внимание, что геометрические свойства верхнего уровня могут работать немного иначе, если в HTML нет <!DOCTYPE HTML>. Возможны странности.
В современном HTML мы всегда должны указывать DOCTYPE.

# Ширина/высота документа

Теоретически, т.к. корневым элементом документа является documentElement, и он включает в себя всё содержимое, мы можем получить полный размер документа как documentElement.scrollWidth/scrollHeight.

Но именно на этом элементе, для страницы в целом, эти свойства работают не так, как предполагается. В Chrome/Safari/Opera, если нет прокрутки, то documentElement.scrollHeight может быть даже меньше, чем documentElement.clientHeight! С точки зрения элемента это невозможная ситуация.

Чтобы надёжно получить полную высоту документа, нам следует взять максимальное из этих свойств:

```
let scrollHeight = Math.max(
  document.body.scrollHeight, document.documentElement.scrollHeight,
  document.body.offsetHeight, document.documentElement.offsetHeight,
  document.body.clientHeight, document.documentElement.clientHeight
);

alert('Полная высота документа с прокручиваемой частью: ' + scrollHeight);
```

Почему? Лучше не спрашивайте. Эти несоответствия идут с древних времён. Глубокой логики здесь нет.

# Получение текущей прокрутки

Обычные элементы хранят текущее состояние прокрутки в ***elem.scrollLeft/scrollTop***.

Что же со страницей? В большинстве браузеров мы можем обратиться к ***documentElement.scrollLeft/Top***, за исключением основанных на старом WebKit (Safari), где есть ошибка (5991), и там нужно использовать ***document.body*** вместо ***document.documentElement***.

К счастью, нам совсем не обязательно запоминать эти особенности, потому что текущую прокрутку можно прочитать из свойств ***window.pageXOffset/pageYOffset***:

Эти свойства доступны только для чтения.

В качестве свойств объекта window также доступны ***scrollX*** и ***scrollY***

По историческим причинам существует два аналога ***window.pageXOffset*** и ***window.pageYOffset***:

- ***window.pageXOffset*** – то же самое, что и ***window.scrollX***.
- ***window.pageYOffse***t – то же самое, что и ***window.scrollY***.

# Прокрутка: scrollTo, scrollBy

Обычные элементы можно прокручивать, изменяя scrollTop/scrollLeft.

Мы можем сделать то же самое для страницы в целом, используя document.documentElement.scrollTop/Left (кроме основанных на старом WebKit (Safari), где, как сказано выше, document.body.scrollTop/Left).

Есть и другие способы, в которых подобных несовместимостей нет: специальные методы window.scrollBy(x,y) и window.scrollTo(pageX,pageY).

- Метод scrollBy(x,y) прокручивает страницу относительно её текущего положения. Например, scrollBy(0,10) прокручивает страницу на 10px вниз.
- Метод scrollTo(pageX,pageY) прокручивает страницу на абсолютные координаты (pageX,pageY). То есть, чтобы левый-верхний угол видимой части страницы имел данные координаты относительно левого верхнего угла документа. Это всё равно, что поставить scrollLeft/scrollTop. Для прокрутки в самое начало мы можем использовать scrollTo(0,0).

В обоих методах вместо координат также может использоваться объект options, как аргумент: options поддерживает три свойства:

- top – то же самое, что y/pageY
- left – то же самое, что x/pageX
- behavior – определяет, каким образом будет прокручиваться страница:
  - "smooth" – плавно (не поддерживается в IE и в старых версиях Safari)
  - "instant" – мгновенно
  - "auto" – определяется браузером (зависит от CSS-свойства scroll-behavior)

```
window.scrollTo({
  top: 100,
  left: 0,
  behavior: "smooth"
});
```

# scrollIntoView

```
this.scrollIntoView({
  behavior: "smooth",
  block: "end",
  inline: "nearest"
});
```

Вызов elem.scrollIntoView(top) прокручивает страницу, чтобы elem оказался вверху. У него есть один аргумент:

- если top=true (по умолчанию), то страница будет прокручена, чтобы elem появился в верхней части окна. Верхний край элемента совмещен с верхней частью окна.
- если top=false, то страница будет прокручена, чтобы elem появился внизу. Нижний край элемента будет совмещен с нижним краем окна.

Как и scrollTo/scrollBy, scrollIntoView также принимает объект options как аргумент (он немного отличается):

- behavior – анимация прокрутки (smooth, instant, auto)
- block – вертикальное выравнивание (start, center, end, nearest). Значение по умолчанию: start
- inline – горизонтальное выравнивание (start, center, end, nearest). Значение по умолчанию: nearest

# Запретить прокрутку

Иногда нам нужно сделать документ «непрокручиваемым». Например, при показе большого диалогового окна над документом – чтобы посетитель мог прокручивать это окно, но не документ.

Чтобы запретить прокрутку страницы, достаточно установить document.body.style.overflow = "hidden".

Аналогичным образом мы можем «заморозить» прокрутку для других элементов, а не только для document.body.

Недостатком этого способа является то, что сама полоса прокрутки исчезает. Если она занимала некоторую ширину, то теперь эта ширина освободится, и содержимое страницы расширится, текст «прыгнет», заняв освободившееся место.

Это выглядит немного странно, но это можно обойти, если сравнить clientWidth до и после остановки, и если clientWidth увеличится (значит полоса прокрутки исчезла), то добавить padding в document.body вместо полосы прокрутки, чтобы оставить ширину содержимого прежней.

# Координаты

- Координаты относительно окна: getBoundingClientRect
- elementFromPoint(x, y)
- Применение для fixed позиционирования
- Координаты относительно документа

Относительно окна браузера – как position:fixed, отсчёт идёт от верхнего левого угла окна.

мы будем обозначать эти координаты как clientX/clientY, причина выбора таких имён будет ясна позже, когда мы изучим свойства событий.

Относительно документа – как position:absolute на уровне документа, отсчёт идёт от верхнего левого угла документа.

мы будем обозначать эти координаты как pageX/pageY.

При прокрутке документа:

- pageY – координата точки относительно документа осталась без изменений, так как отсчет по-прежнему ведётся от верхней границы документа (сейчас она прокручена наверх).
- clientY – координата точки относительно окна изменилась (стрелка на рисунке стала короче), так как точка стала ближе к верхней границе окна.

# Координаты относительно окна: getBoundingClientRect

Метод elem.getBoundingClientRect() возвращает координаты в контексте окна для минимального по размеру прямоугольника, который заключает в себе элемент elem, в виде объекта встроенного класса DOMRect.

Основные свойства объекта типа DOMRect:

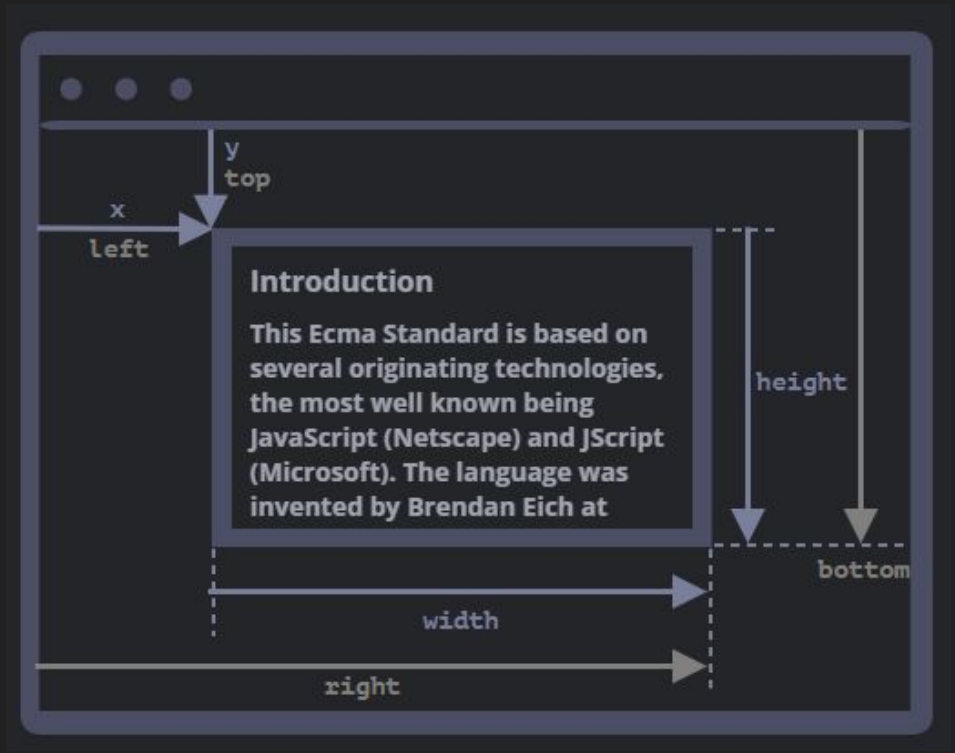
- x/y – X/Y-координаты начала прямоугольника относительно окна,
- width/height – ширина/высота прямоугольника (могут быть отрицательными).

Дополнительные, «зависимые», свойства:

- top/bottom – Y-координата верхней/нижней границы прямоугольника,
- left/right – X-координата левой/правой границы прямоугольника.

Координаты могут считаться с десятичной частью, например 10.5. Это нормально, ведь браузер использует дроби в своих внутренних вычислениях. Мы не обязаны округлять значения при установке style.left/top.

Координаты могут быть отрицательными. Например, если страница прокручена так, что элемент elem ушёл вверх за пределы окна, то вызов elem.getBoundingClientRect().top вернет отрицательное значение.



# особенности getBoundingClientRect

**Internet Explorer и Edge: не поддерживают x/y**

Internet Explorer и Edge не поддерживают свойства x/y по историческим причинам.

Таким образом, мы можем либо сделать полифил (добавив соответствующие геттеры в DomRect.prototype), либо использовать top/left, так как это всегда одно и то же при положительных width/height, в частности – в результате вызова elem.getBoundingClientRect().

**Координаты right/bottom отличаются от одноименных CSS-свойств**

Есть очевидное сходство между координатами относительно окна и CSS position:fixed.

Но в CSS свойство right означает расстояние от правого края, и свойство bottom означает расстояние от нижнего края окна браузера.

Если взглянуть на картинку выше, то видно, что в JavaScript это не так. Все координаты в контексте окна считаются от верхнего левого угла, включая right/bottom.

# elementFromPoint(x, y)

Вызов document.elementFromPoint(x, y) возвращает самый глубоко вложенный элемент в окне, находящийся по координатам (x, y).

Поскольку используются координаты в контексте окна, то элемент может быть разным, в зависимости от того, какая сейчас прокрутка.

Для координат за пределами окна метод elementFromPoint возвращает null

Метод document.elementFromPoint(x,y) работает, только если координаты (x,y) относятся к видимой части содержимого окна.

Если любая из координат представляет собой отрицательное число или превышает размеры окна, то возвращается null.

```
let elem = document.elementFromPoint(x, y);
```

```
let centerX = document.documentElement.clientWidth / 2;
let centerY = document.documentElement.clientHeight / 2;

let elem = document.elementFromPoint(centerX, centerY);

elem.style.background = "red";
alert(elem.tagName);
```

```
let elem = document.elementFromPoint(x, y);
// если координаты ведут за пределы окна, то elem = null
elem.style.background = ''; // Ошибка!
```

# Применение для fixed позиционирования

Чаще всего нам нужны координаты для позиционирования чего-либо.

Чтобы показать что-то около нужного элемента, мы можем вызвать getBoundingClientRect, чтобы получить его координаты, а затем использовать CSS-свойство position вместе с left/top (или right/bottom).

Например, функция createMessageUnder(elem, html) ниже показывает сообщение под элементом elem:

Код можно изменить, чтобы показывать сообщение слева, справа, снизу, применять к нему CSS-анимации и так далее. Это просто, так как в нашем распоряжении имеются все координаты и размеры элемента.

Но обратите внимание на одну важную деталь: при прокрутке страницы сообщение уплывает от кнопки.

Причина весьма очевидна: сообщение позиционируется с помощью position:fixed, поэтому оно остаётся всегда на том же самом месте в окне при прокрутке страницы.

Чтобы изменить это, нам нужно использовать другую систему координат, где сообщение позиционировалось бы относительно документа, и свойство position:absolute.

```
let elem = document.getElementById("coords-show-mark");

function createMessageUnder(elem, html) {
  // создаём элемент, который будет содержать сообщение
  let message = document.createElement('div');
  // для стилей лучше было бы использовать css-класс здесь
  message.style.cssText = "position:fixed; color: red";

  // устанавливаем координаты элементу, не забываем про "px"!
  let coords = elem.getBoundingClientRect();

  message.style.left = coords.left + "px";
  message.style.top = coords.bottom + "px";

  message.innerHTML = html;

  return message;
}

// Использование:
// добавим сообщение на страницу на 5 секунд
let message = createMessageUnder(elem, 'Hello, world!');
document.body.append(message);
setTimeout(() => message.remove(), 5000);
```

# Координаты относительно документа

В такой системе координат отсчёт ведётся от левого верхнего угла документа, не окна.

В CSS координаты относительно окна браузера соответствуют свойству position:fixed, а координаты относительно документа – свойству position:absolute на самом верхнем уровне вложенности.

Мы можем воспользоваться свойствами position:absolute и top/left, чтобы привязать что-нибудь к конкретному месту в документе. При этом прокрутка страницы не имеет значения. Но сначала нужно получить верные координаты.

Не существует стандартного метода, который возвращал бы координаты элемента относительно документа, но мы можем написать его сами.

Две системы координат связаны следующими формулами:

- pageY = clientY + высота вертикально прокрученной части документа.
- pageX = clientX + ширина горизонтально прокрученной части документа.

Функция getCoords(elem) берёт координаты в контексте окна с помощью elem.getBoundingClientRect() и добавляет к ним значение соответствующей прокрутки:

Если бы в примере выше мы использовали её вместе с position:absolute, то при прокрутке сообщение оставалось бы рядом с элементом.

Модифицированная функция createMessageUnder:

```
// получаем координаты элемента в контексте документа
function getCoords(elem) {
  let box = elem.getBoundingClientRect();

  return {
    top: box.top + window.pageYOffset,
    right: box.right + window.pageXOffset,
    bottom: box.bottom + window.pageYOffset,
    left: box.left + window.pageXOffset
  };
}
```

```
function createMessageUnder(elem, html) {
  let message = document.createElement('div');
  message.style.cssText = "position:absolute; color: red";

  let coords = getCoords(elem);

  message.style.left = coords.left + "px";
  message.style.top = coords.bottom + "px";

  message.innerHTML = html;

  return message;
}
```

# Итого

Любая точка на странице имеет координаты:

- Относительно окна браузера – elem.getBoundingClientRect().
- Относительно документа – elem.getBoundingClientRect() плюс текущая прокрутка страницы.

Координаты в контексте окна подходят для использования с position:fixed, а координаты относительно документа – для использования с position:absolute.

Каждая из систем координат имеет свои преимущества и недостатки. Иногда будет лучше применить одну, а иногда – другую, как это и происходит с позиционированием в CSS, где мы выбираем между absolute и fixed.

# canvas

<canvas> — это HTML-элемент, который используется для рисования графики с помощью JavaScript. Он позволяет создавать динамические изображения, анимации, графики и другие визуальные элементы непосредственно на веб-странице.

Основные моменты про <canvas>:

- Создание Canvas
- Получение контекста рисования
- Рисование на Canvas

Использование <canvas> открывает широкие возможности для создания интерактивных и динамических графических элементов на веб-страницах, что делает его мощным инструментом в арсенале веб-разработчика.

Чтобы использовать <canvas>, сначала его нужно создать в HTML:

```
<canvas id="myCanvas" width="500" height="400"></canvas>
```

Для рисования на <canvas> нужно получить контекст рисования. Обычно используется двумерный контекст (2d):

```
const canvas = document.getElementById('myCanvas');
const ctx = canvas.getContext('2d');
```

После получения контекста можно начинать рисовать. Например, нарисуем прямоугольник:

```
// Задаем цвет заливки
ctx.fillStyle = 'green';
// Рисуем прямоугольник
ctx.fillRect(10, 10, 150, 100);
```

# Примеры рисования

Рисование линий:

```
ctx.beginPath();        // Начинаем новый путь
ctx.moveTo(50, 50);     // Начальная точка
ctx.lineTo(200, 50);    // Конечная точка
ctx.stroke();           // Рисуем линию
```

Рисование окружностей:

```
ctx.beginPath();
ctx.arc(150, 75, 50, 0, 2 * Math.PI); // x, y, радиус, начальный угол, конечный угол
ctx.stroke(); // Рисуем окружность
```

Рисование текста:

```
ctx.font = '30px Arial';
ctx.fillText('Hello World', 10, 50);
```

# Полезные свойства и методы контекста 2d

fillStyle и strokeStyle — задают цвет заливки и обводки соответственно.

fillRect(x, y, width, height) — рисует прямоугольник.

strokeRect(x, y, width, height) — рисует контур прямоугольника.

clearRect(x, y, width, height) — очищает прямоугольную область.

beginPath() — начинает новый путь (для рисования).

moveTo(x, y) — перемещает перо в указанную точку.

lineTo(x, y) — рисует линию до указанной точки.

arc(x, y, radius, startAngle, endAngle) — рисует дугу или окружность.

fillText(text, x, y) — рисует заполненный текст.

strokeText(text, x, y) — рисует контур текста.

# Ресурсы

Размеры и прокрутка элементов - ТЫК

Размеры и прокрутка окна - ТЫК

Координаты - ТЫК

<canvas> элемент MDN документация - ТЫК

Canvas API MDN документация - ТЫК