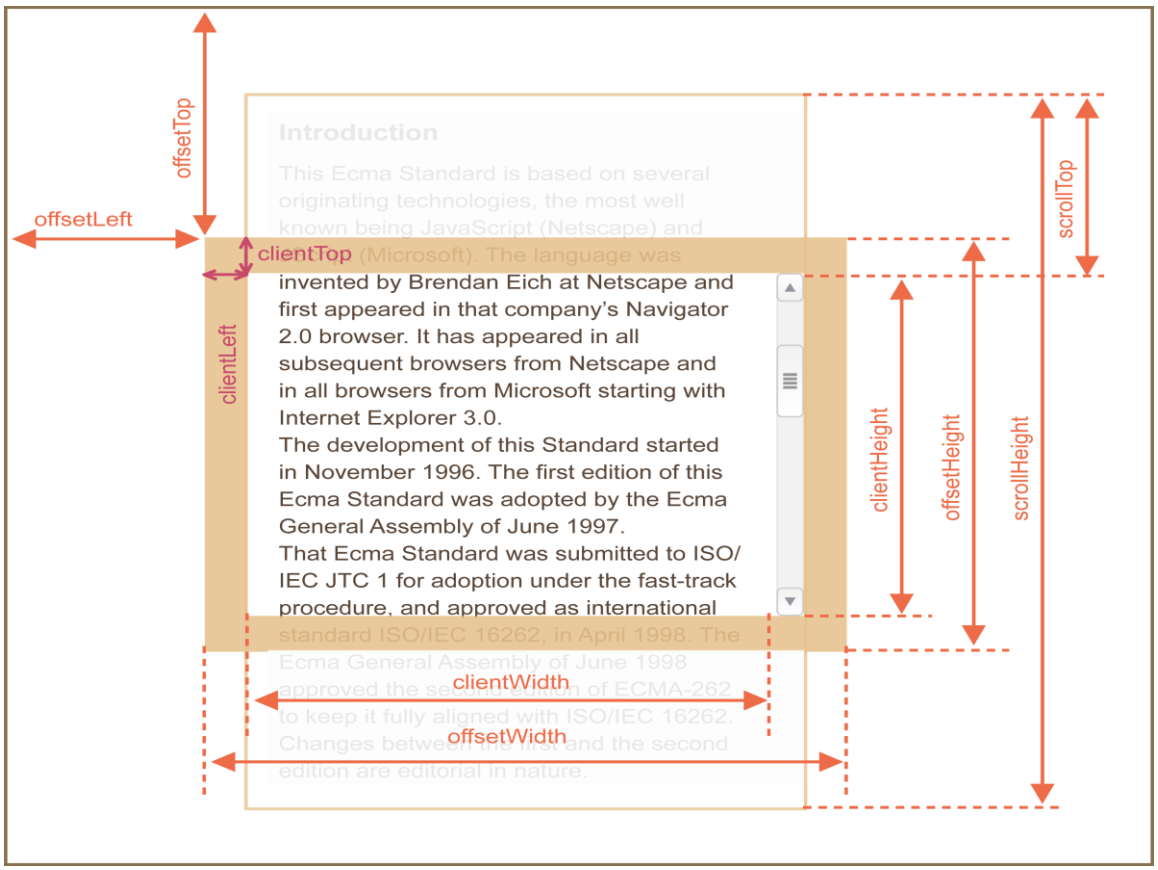
Метрики элемента



Значениями свойств являются **числа**, подразумевается, что они **в пикселях**.

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.
В примере ниже внутренний имеет элемент <main> в качестве offsetParent, а свойства offsetLeft/offsetTop являются сдвигами относительно верхнего левого угла (180):

Существует несколько ситуаций, когда **offsetParent = 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

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

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 = null, а offsetWidth и offsetHeight = 0, когда мы создали элемент, но ещё не вставили его в документ, или если у элемента (или у его родителя) display:none. Мы можем использовать это, чтобы делать проверку на видимость:

```
function isHidden(elem) {
  return !elem.offsetWidth && !elem.offsetHeight;
}
```

Заметим, что функция isHidden также вернёт true для элементов, которые в принципе показываются, но их размеры равны нулю (например, пустые).

clientTop/Left

Пойдём дальше. Внутри элемента у нас рамки (border). Для них есть свойства-метрики **clientTop** и **clientLeft**. …Но на самом деле эти свойства – вовсе не ширины рамок, а отступы внутренней части элемента от внешней. В чём же разница? Она

возникает, когда документ располагается справа налево (операционная система на арабском языке или иврите). Полоса прокрутки в этом случае находится слева, и тогда свойство clientLeft включает в себя ещё и ширину полосы прокрутки.

clientWidth/Height

Эти свойства – размер области внутри рамок элемента. Они включают в себя ширину области содержимого **вместе с внутренними отступами padding**, но без прокрутки. 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.
Эти свойства можно использовать, чтобы «распахнуть» элемент на всю ширину/высоту. Таким кодом:

```
// распахнуть элемент на всю высоту
element.style.height = `${element.scrollHeight}px`;
```

scrollLeft/scrollTop

Свойства scrollLeft/scrollTop – ширина/высота невидимой, прокрученной в данный момент, части элемента слева и сверху. В отличие от большинства свойств, которые доступны только для чтения, значения scrollLeft/scrollTop **можно изменять**, и браузер выполнит прокрутку элемента.

Не стоит брать width/height из CSS! Как мы знаем из главы Стили и классы, CSS-высоту и ширину можно извлечь, используя getComputedStyle. Так почему бы не получать, к примеру, ширину элемента при помощи getComputedStyle, вот так?

```
let elem = document.body;
alert( getComputedStyle(elem).width ); // показывает CSS-ширину elem
```

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

- 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.
- Во-вторых, в CSS свойства width/height могут быть равны auto, например, для инлайнового элемента:

Есть и ещё одна причина: полоса прокрутки. Бывает, без полосы прокрутки код работает прекрасно, но стоит ей появиться, как начинают проявляться баги. Так происходит потому, что полоса прокрутки «отъедает» место от области внутреннего содержимого в некоторых браузерах. Таким образом, реальная ширина содержимого меньше CSS-ширины. Как раз это и учитывают свойства clientWidth/clientHeight. …Но с getComputedStyle(elem).width ситуация иная. Некоторые браузеры (например, Chrome) возвращают реальную внутреннюю ширину с вычетом ширины полосы прокрутки, а некоторые (например, Firefox) – именно CSS-свойство (игнорируя полосу прокрутки). Эти кроссбраузерные отличия – ещё один повод не использовать getComputedStyle, а использовать свойства-метрики.

Размеры и прокрутка окна

Ширина/высота окна

Чтобы получить ширину/высоту окна, можно взять свойства document.documentElement.clientHeight/Width.
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.

Не window.innerWidth/Height ! 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 неё (за её вычетом). Иными словами, они возвращают высоту/ширину видимой части документа, доступной для содержимого.
А window.innerWidth/innerHeight включают в себя полосу прокрутки. Если полоса прокрутки занимает некоторое место, то эти две строки выведут разные значения:

```
alert( window.innerWidth ); // полная ширина окна
alert( document.documentElement.clientWidth ); // ширина окна за вычетом полосы прокрутки
```

В большинстве случаев нам нужна *доступная* ширина окна: для рисования или позиционирования. Полоса прокрутки «отъедает» её часть.

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);
```

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:

```
alert('Текущая прокрутка сверху: ' + window.pageYOffset);
alert('Текущая прокрутка слева: ' + window.pageXOffset);
```

Эти свойства доступны только для чтения.

Прокрутка страницы: scrollTo, scrollBy, scrollIntoView

Важно: Для прокрутки страницы из JavaScript её DOM должен быть полностью построен. Например, если мы попытаемся прокрутить страницу из скрипта в <head>, это не сработает. 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).

Эти методы одинаково работают для всех браузеров.

Также используется другой синтаксис для scrollTo (не работает в safari):

```
window.scrollTo({
  top: 500,
  left: 0,
  behavior: "smooth"  // instant, auto (по умолчанию)
 });
```

Для полноты картины давайте рассмотрим ещё один метод: elem.scrollIntoView(top). 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 совмещён с верхней частью окна. если top = false, то страница будет прокручена, чтобы elem появился внизу. Нижний край элемента будет совмещён с нижним краем окна.

Есть опции (не работает в safari):

```
elem.scrollIntoView({
  block: “center”,  // start, center (по умолчанию), end, nearest – позиционирование по вертикали
  inline: “nearest”, // start, center, end, nearest(по умолчанию) – позиционирование по горизонтали
  behavior: “smooth”
}).
```

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".

document.body.style.overflow = "" – возобновление прокрутки.

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

Большинство соответствующих методов JavaScript работают в одной из двух указанных ниже систем координат:

- **Относительно окна браузера** – как position:fixed, отсчёт идёт от верхнего левого угла окна. Мы будем обозначать эти координаты как clientX/clientY,
- **Относительно документа** – как position:absolute на уровне документа, отсчёт идёт от верхнего левого угла документа. Мы будем обозначать эти координаты как pageX/pageY.

Когда страница полностью прокручена в самое начало, то верхний левый угол окна совпадает с левым верхним углом документа, при этом обе этих системы координат тоже совпадают. Но если происходит прокрутка, то координаты элементов в контексте окна меняются, так как они двигаются, но в то же время их координаты относительно документа остаются такими же.

Координаты относительно окна: getBoundingClientRect

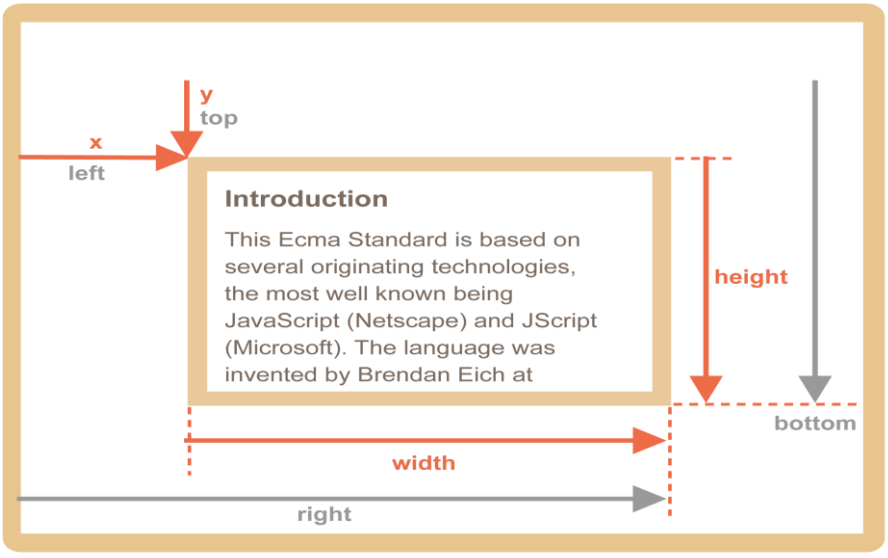
Метод elem.getBoundingClientRect() возвращает координаты в контексте окна для минимального по размеру прямоугольника, который заключает в себе элемент elem, в виде объекта встроенного класса DOMRect.
Основные свойства объекта типа DOMRect:

- **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.

Вот картинка с результатами вызова elem.getBoundingClientRect():



Заметим, что left = x, top = y, right = x + width, bottom = y + height (почти всегда, за исключением отрицательных значений ширины и высоты). Дополнительные зависимые свойства существуют лишь для удобства.
Координаты могут считаться с десятичной частью. Координаты могут быть отрицательными.
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,
Координаты **right/bottom отличаются от одноимённых 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)

Если мы хотим узнать – какой объект находится на заданных координатах:
Вызов document.elementFromPoint(x, y) возвращает самый глубоко вложенный элемент **в окне**, находящийся по координатам (x, y). Синтаксис:

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

Например, код ниже выделяет с помощью стилей и выводит имя тега элемента, который сейчас в центре окна браузера:

```
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);
```

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**.

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 элемента, а затем использовать CSS-свойство position вместе с **left/top (или right/bottom).**
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.

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.
Две системы координат связаны следующими формулами:

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

Например, функция createMessageUnder(elem, html) ниже показывает сообщение под элементом elem: Функция getCoords(elem) берёт координаты в контексте окна с помощью elem.getBoundingClientRect() и добавляет к ним значение соответствующей прокрутки:

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

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