

Размеры и прокрутка элементов

Существует множество JavaScript-свойств, которые позволяют считывать информацию об элементе: ширину, высоту и другие геометрические характеристики. В этой главе мы будем называть их «метрики».

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

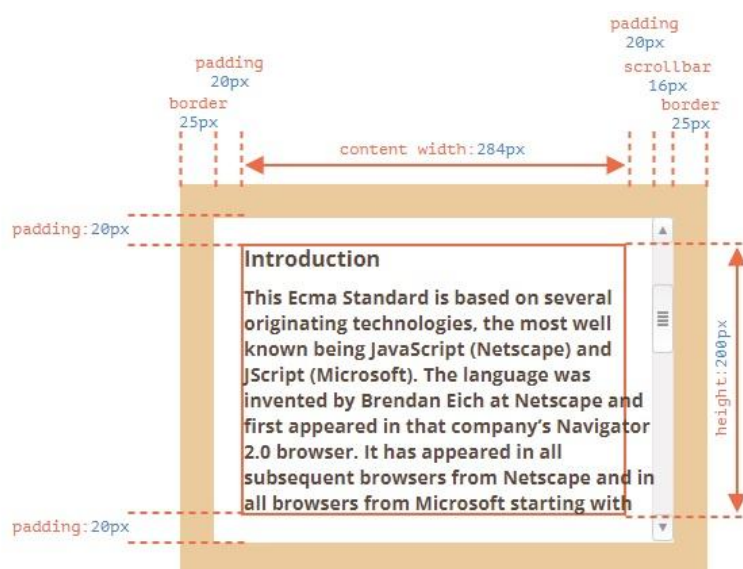
Простой пример

В качестве простого примера демонстрации свойств мы будем использовать следующий элемент:

```
<div id="example">
  ...Текст...
</div>
<style>
  #example {
    width: 300px;
    height: 200px;
    border: 25px solid #E8C48F;
    padding: 20px;
    overflow: auto;
  }
</style>
```

У элемента есть рамка (border), внутренний отступ (padding) и прокрутка. Полный набор характеристик. Обратите внимание, тут нет внешних отступов (margin), потому что они не являются частью элемента, для них нет особых JavaScript-свойств.

Результат выглядит так:



Внимание, полоса прокрутки

В иллюстрации выше намеренно продемонстрирован самый сложный и полный случай, когда у элемента есть ещё и полоса прокрутки. Некоторые браузеры (не все) отбирают место для неё, забирая его у области, отведённой для содержимого (помечена как «content width» выше).

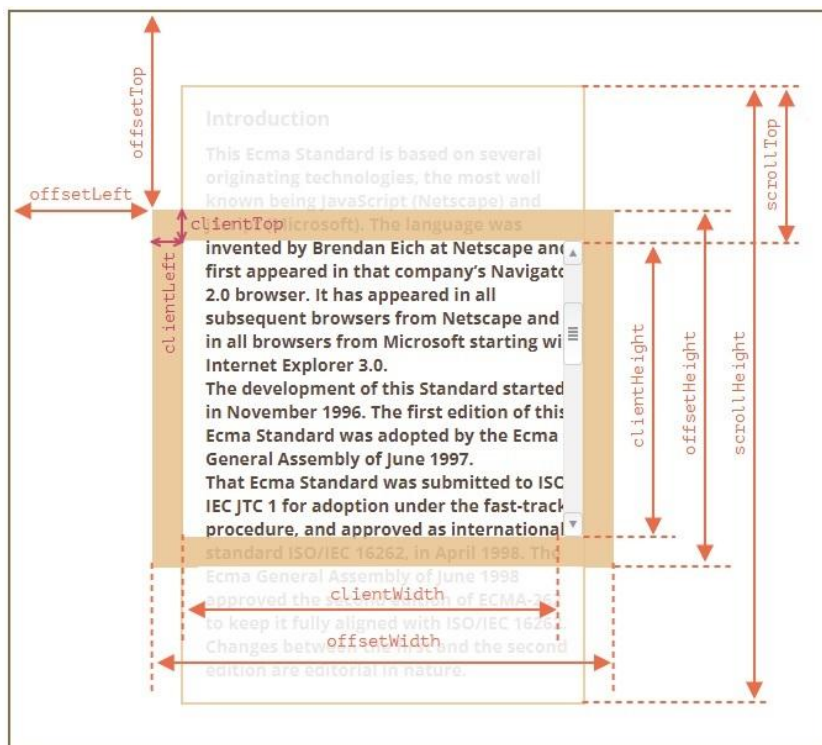
Таким образом, без учёта полосы прокрутки ширина области содержимого (content width) будет 300px, но если предположить, что ширина полосы прокрутки равна 16px (её точное значение зависит от устройства и браузера), тогда остаётся только $300 - 16 = 284\text{px}$, и мы должны это учитывать. Вот почему примеры в этой главе даны с полосой прокрутки. Без неё некоторые вычисления будут проще.

Область padding-bottom (нижний внутренний отступ) может быть заполнена текстом

Нижние внутренние отступы padding-bottom изображены пустыми на наших иллюстрациях, но если элемент содержит много текста, то он будет перекрывать padding-bottom, это нормально.

Метрики

Вот общая картина с геометрическими свойствами:



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

Давайте начнём исследовать, начиная снаружи элемента.

offsetParent, offsetLeft/Top

Эти свойства редко используются, но так как они являются «самыми внешними» метриками, мы начнём с них.

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

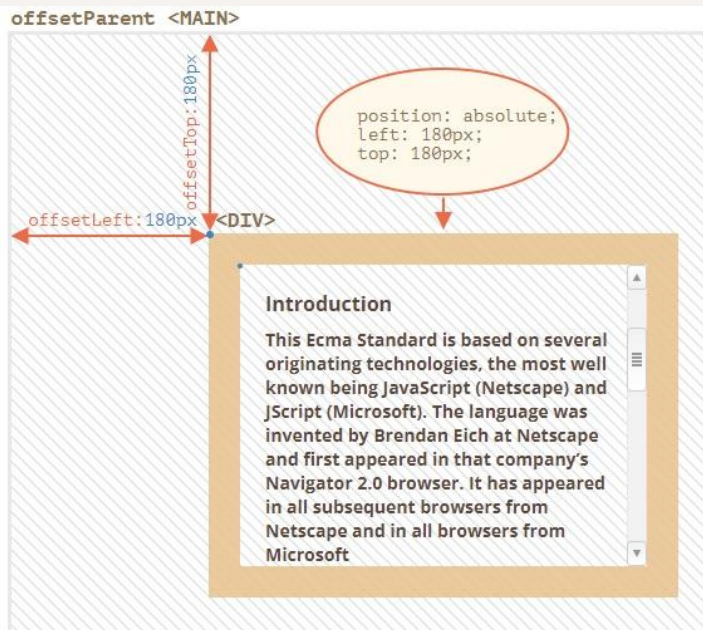
То есть, ближайший предок, который удовлетворяет следующим условиям:

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

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

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

```
<main style="position: relative" id="main">
  <article>
    <div id="example" style="position: absolute; left: 180px; top:
180px">...</div>
  </article>
</main>
<script>
  alert(example.offsetParent.id); // main
  alert(example.offsetLeft); // 180 (обратите внимание: число, а не строка
"180px")
  alert(example.offsetTop); // 180
</script>
```



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

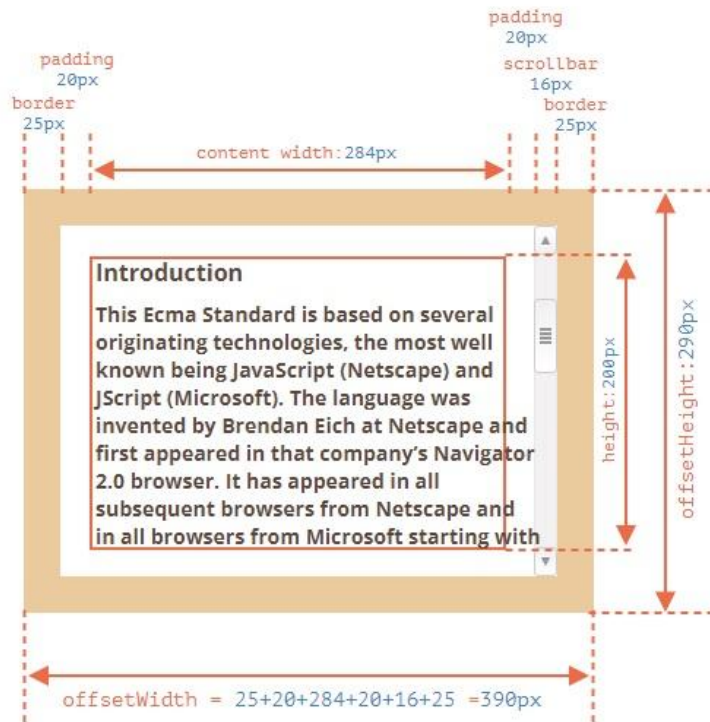
1. Для скрытых элементов (с CSS-свойством `display: none` или когда его нет в документе).
2. Для элементов `<body>` и `<html>`.

3. 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.



Для нашего элемента:

- `offsetWidth` = 390 – внешняя ширина блока, её можно получить сложением CSS-ширины (300px), внутренних отступов ($2 * 20\text{px}$) и рамок ($2 * 25\text{px}$).
- `offsetHeight` = 290 – внешняя высота блока.

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`.

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 `<div>`).

clientTop/Left

Пойдём дальше. 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.

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.

В этом случае `clientLeft` будет равно 25, но с прокруткой – $25 + 16 = 41$.

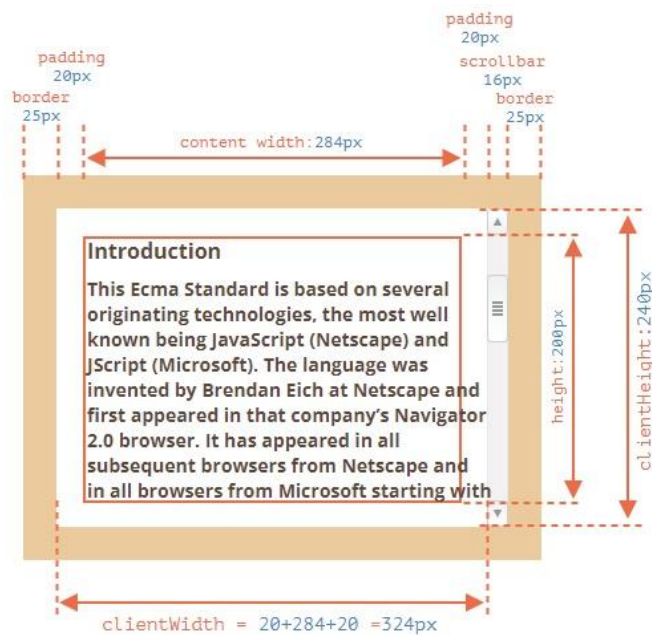
Вот соответствующий пример на иврите:



clientWidth/Height

Эти свойства – размер области внутри рамок элемента.

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

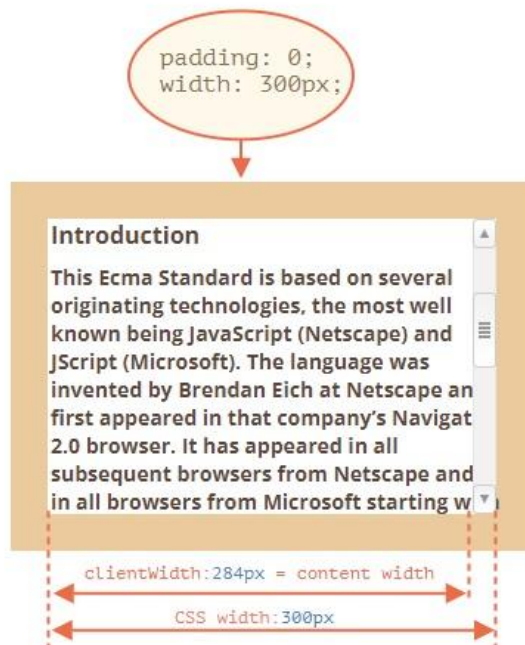


На рисунке выше посмотрим вначале на высоту `clientHeight`.

Горизонтальной прокрутки нет, так что это в точности то, что внутри рамок: CSS-высота 200px плюс верхние и нижние внутренние отступы ($2 * 20px$), итого 240px.

Теперь `clientWidth` – ширина содержимого здесь равна не 300px, а 284px, т.к. 16px отведено для полосы прокрутки. Таким образом: 284px плюс левый и правый отступы – всего 324px.

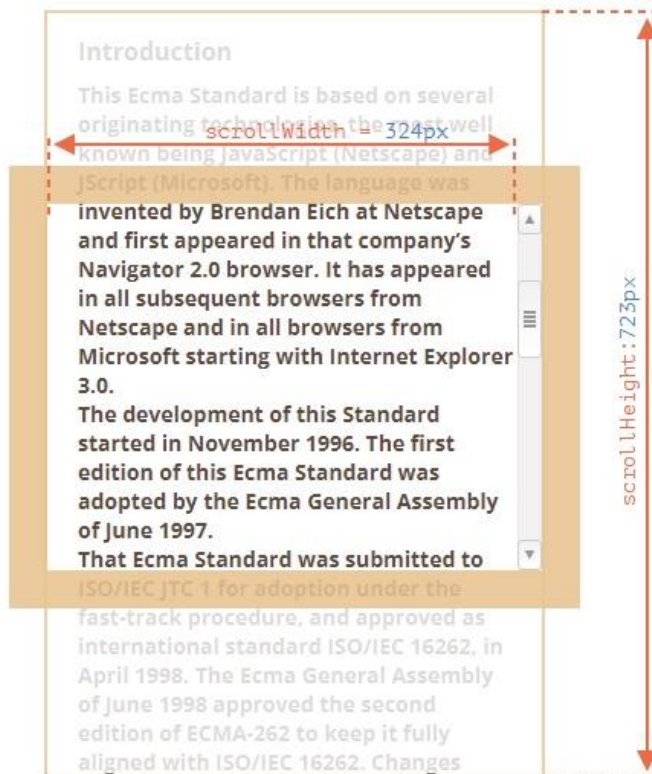
Если нет внутренних отступов padding, то `clientWidth/Height` в точности равны размеру области содержимого внутри рамок и полосы прокрутки (если она есть).



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.



На рисунке выше:

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

Эти свойства можно использовать, чтобы «распахнуть» элемент на всю ширину/высоту.

Таким кодом:

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

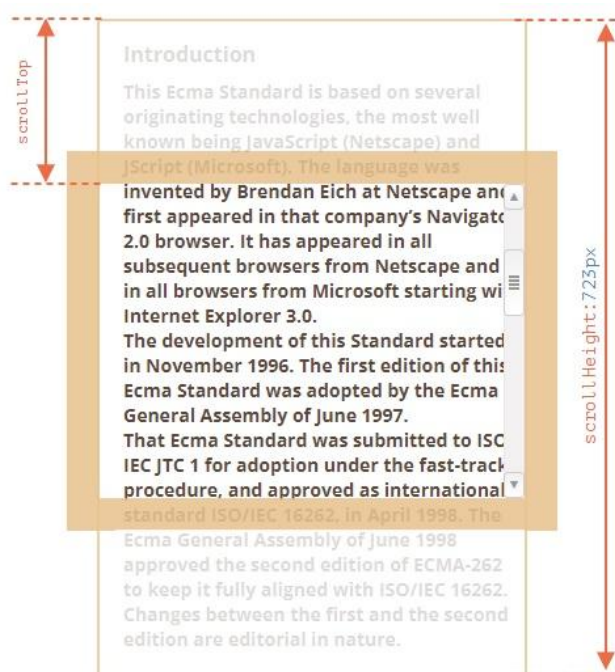
Нажмите на кнопку, чтобы распахнуть элемент:



scrollTop/scrollLeft

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

Следующая иллюстрация показывает значения `scrollHeight` и `scrollTop` для блока с вертикальной прокруткой.

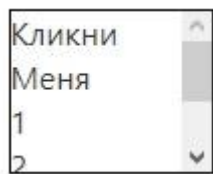


Другими словами, свойство `scrollTop` – это «сколько уже прокручено вверх».

Свойства `scrollTop/scrollLeft` можно изменять

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

При клике на следующий элемент будет выполняться код `elem.scrollTop += 10`. Поэтому он будет прокручиваться на 10px вниз.



Установка значения `scrollTop` на 0 или `Infinity` прокрутит элемент в самый верх/низ соответственно.

Не стоит брать width/height из CSS

Мы рассмотрели метрики, которые есть у DOM-элементов, и которые можно использовать для получения различных высот, ширин и прочих расстояний.

Но 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:

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

```
3. <span id="elem">Привет!</span>
4.
5. <script>
6.   alert( getComputedStyle(elem).width ); // auto
   </script>
```

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.

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.

Если ваш браузер показывает полосу прокрутки (например, под Windows почти все браузеры так делают), то вы можете протестировать это сами, нажав на кнопку в ифрейме ниже.



У элемента с текстом в стилях указано CSS-свойство `width:300px`.

На ОС Windows браузеры Firefox, Chrome и Edge резервируют место для полосы прокрутки. Но Firefox отображает 300px, в то время как Chrome и Edge – меньше. Это из-за того, что Firefox возвращает именно CSS-ширину, а остальные браузеры – «реальную» ширину за вычетом прокрутки.

Обратите внимание: описанные различия касаются только чтения свойства `getComputedStyle(...).width` из JavaScript, визуальное отображение корректно в обоих случаях.

Итого

У элементов есть следующие геометрические свойства (метрики):

- `offsetParent` – ближайший CSS-позиционированный родитель или ближайший `td`, `th`, `table`, `body`.
- `offsetLeft/offsetTop` – позиция в пикселях верхнего левого угла относительно `offsetParent`.
- `offsetWidth/offsetHeight` – «внешняя» ширина/высота элемента, включая рамки.
- `clientLeft/clientTop` – расстояние от верхнего левого внешнего угла до внутреннего. Для операционных систем с ориентацией слева-направо эти свойства равны ширинам левой/верхней рамки. Если язык ОС таков, что ориентация справа налево, так что вертикальная полоса прокрутки находится не справа, а слева, то `clientLeft` включает в своё значение её ширину.
- `clientWidth/clientHeight` – ширина/высота содержимого вместе с внутренними отступами `padding`, но без полосы прокрутки.
- `scrollWidth/scrollHeight` – ширины/высота содержимого, аналогично `clientWidth/Height`, но учитывают прокрученную, невидимую область элемента.
- `scrollLeft/scrollTop` – ширина/высота прокрученной сверху части элемента, считается от верхнего левого угла.

Все свойства доступны только для чтения, кроме `scrollLeft/scrollTop`, изменение которых заставляет браузер прокручивать элемент.

Задачи

Найти размер прокрутки снизу

важность: 5

Свойство `elem.scrollTop` содержит размер прокрученной области при отсчёте сверху. А как подсчитать размер прокрутки снизу (назовём его `scrollBottom`)?

Напишите соответствующее выражение для произвольного элемента `elem`.

P.S. Проверьте: если прокрутки нет вообще или элемент полностью прокручен – оно должно давать 0.

Узнать ширину полосы прокрутки

важность: 3

Напишите код, который возвращает ширину стандартной полосы прокрутки.

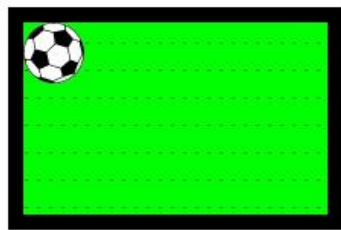
Для Windows она обычно колеблется от 12px до 20px. Если браузер не выделяет место под полосу прокрутки (так тоже бывает, она может быть прозрачной над текстом), тогда значение может быть 0px.

P.S. Ваш код должен работать в любом HTML-документе, независимо от его содержимого.

Поместите мяч в центр поля

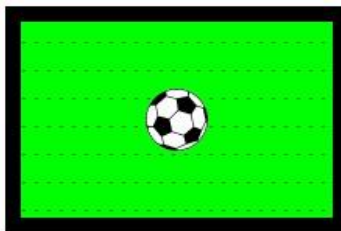
важность: 5

Исходный документ выглядит так:



Каковы координаты центра поля?

Вычислите их и используйте, чтобы поместить мяч в центр поля:



- Элемент должен позиционироваться за счёт JavaScript, а не CSS.
- Код должен работать с любым размером мяча (10, 20, 30 пикселей) и любым размером поля без привязки к исходным значениям.

P.S. Да, центрирование можно сделать при помощи чистого CSS, но задача именно на JavaScript. Далее будут другие темы и более сложные ситуации, когда JavaScript будет уже точно необходим, это – своего рода «разминка».

В чём отличие CSS-свойств width и clientWidth

важность: 5

В чём отличие между `getComputedStyle(elem).width` и `elem.clientWidth`?

Укажите хотя бы 3 отличия, лучше – больше.