ПОДІЇ

Всі зміни, які відбуваються на Web-сторінки, пов'язані з роботою браузера або маніпуляціями користувача з клавішами миші або клавіатури, називаються подіями. Для вказівки дій, які необхідно зробити в зв'язку з появою того або іншої події, використовуються обробники подій.

**Типи подій**

Можна виділити декілька груп подій, у залежності від того як вони генеруються. Розглянемо деякі з них

**Події документу**

load              Закінчено завантаження документа (Frame, Image, Layer)

unload                         Користувач залишає сторінку, наприклад, завантажуючи інший документ (Frame, window)

resize           Користувач змінив розмір вікна (Frame, window )

error             Виникла помилка підчас завантаження ( Image, window)

move            Користувач перемістив вікно (Frame, window)

DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

**Події миші**

click                         Користувач натискає кнопку миші (Button, Checkbox, document, Link, Radio, Reset, Submit , і т.д. – майже всі об'єкти

mousedown              Користувач натиснув кнопку миші. (Button, document, Link )

mouseup      Користувач відпустив конпку миші (Button, document, Link)

dblclick        Кристувач двічі натискає на кнопку миші (Button, Checkbox, document, FileUpload, Hidden, Link, Password, Radio, Reset, Select, Submit )

dragdrop      Користувач перетягує об'єкт у вікно

mousemove              Користувач перемістив курсор миші (Виникає лише при явному заданні обробника).

mouseout     Користувач перемістив курсор за межі об'єкта(Area, Layer, Link)

mouseover   Користувач помістив курсор миші над об'єктом (Area, Layer, Link)

contextmenu – відбувається при виклику констекстного меню (права кнопка миші)

**Події клавіатури**

keydown      Користувач натиснув клавішу (document, Image, Link, Text, Textarea)

keyup      Користувач відпустив клавішу (document, Image, Link, Text, Textarea)

keypress      Користувач натиснув клавішу і відпустив. (document, Image, Link)

**Події елемента**

abort            Користувач зупиняє завантаженнязображення (Image)

blur                          Користувач прибирає фокус із об'єкта  (Button,Checkbox,FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window)

change         Користувач змінює зміст елемента форми (the FileUpload, Select, Text)

error             Виникла помилка підчас завантаження ( Image, window)

focus            Користувач перемістив фокус на об'єкт (Button, Checkbox, FileUpload,

Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, window )

select           Користвуач виділив текст (Text, Textarea)

**Події форми**

reset             Користувач натиснув кнопку Reset у формі (Form)

submit          Користвуач відправив дані з форми (Form)

**События CSS:**

* transitionend – когда CSS-анимация завершена.

=====================================================================================

**ОБРОБКА ПОДІЙ**

Під обробкою подій розуміють виконання деякої команди або ж функції у відповідь на настання події. Обробка подій може здійснювати декількома способами

**Задання обробника з використанням атрибуту елемента HTML**

У цьому способі необхідно у HTML розмітці елемента додати атрибут, який починається з «on» і містить назву події (наприклад: **on**click, **on**change, **on**select,…). Значенням атрибуту може бути або деяка команда або виклик функції. Як завжди значення атрибутів вказуємо у лапках.

|  |  |
| --- | --- |
| Загальна форма | <елемент **on**подія= “виклик\_обробника”> |
| Приклад (вказівка команди) | <input type="button" name="name" value=”Press” onclick="alert(“Hello”)" /> |
| Приклад (виклик функції) | <input type="button" name="name" value=”Press” onclick="myFunc()" /> |

**Задання обробника з використанням властивості елемента**

У цьому способі необхідно використати властивість елемента як об’єкта JavaScript. При цьому ім’я властивості, як і атрибут має вигляд **on**подія. На відміну від атрибутів елементів, які можуть бути вказані як у нижньому так і верхньому регістрах (onclick, ONCLICK, Onclick,onClick), властивості елемента вказують у нижньому регістрі (onclick).

|  |  |
| --- | --- |
| Загальна форма | елемент . **on**подія= функція\_обробника ;  елемент [ “**on**подія ” ]= функція\_обробника ; |
| Приклад | ------------- Елемент в HTML -------------  <input type="button" id="myButton" value=”Press” />  ------------- Встановлення обробника в JavaScript --------------  <script>  //------ Варіант 1 -----  myButton.onclick = myFunc; // myFunc – функція обробник  //------ Варіант 2 -----  myButton["onclick"] = myFunc; // myFunc – функція обробник  </script> |

**Задання і відміна обробника з використанням методів addEventListener та removeEventListener**

У цьому способі для призначення обробника необхідно у методі елемента **addEventListener** вказати назву події та функцію обробник.

|  |  |
| --- | --- |
| Загальна форма | елемент . **addEventListener(** подія, функція\_обробник , фаза );  фаза (true/false) – необов’язковий параметр (за замовчуванням false), який зазначає коли повинен спрацювати обробник (на нисхідному чи висхідному етапі) |
| Приклад | ------------- Елемент в HTML -------------  <input type="button" id="myButton" value=”Press” />  ------------- Встановлення обробника в JavaScript --------------  <script>  myButton.addEventListener("click", myFunc); // myFunc – функція обробник  </script> |

Перевагою такого підходу є те, що ми маємо можливість вказати декілька функцій обробників для однієї події

|  |  |
| --- | --- |
| Приклад | ------------- Елемент в HTML -------------  <input type="button" id="myButton" value=”Press” />  ------------- Встановлення обробників в JavaScript --------------  <script>  myButton.addEventListener("click", myFunc); // призначення обробника myFunc  myButton.addEventListener("click", myFunc2); //призначення ще одного обробника myFunc2  //при виникненні події буде викликано обидві функції  </script> |

Для видалення обробника необхідно використати метод

|  |  |
| --- | --- |
| Приклад | ------------- Елемент в HTML -------------  <input type="button" id="myButton" value=”Press” />  ------------- Встановлення обробників в JavaScript --------------  <script>  myButton.addEventListener("click", myFunc); // призначення обробника myFunc  myButton.addEventListener("click", myFunc2); //призначення ще одного обробника myFunc2    myButton.removeEventListener("click", myFunc2); // видалення обробника myFunc2  //при виникненні події буде викликано тільки функцію myFunc  </script> |

**Відміна події**

Для відміни дії браузера на подію у даному елементі достатньо, щоб функція обробник повернула false або скористатися методом об’єкта події **event.preventDefault()**.

|  |  |
| --- | --- |
| Приклад | ----- Відміити перехід за посиланням при кліку -----  ------------- Елемент в HTML -------------  <a href='Any.html' **onclick='return false;'**>link</a> |

========================================================================

**Значення this при виклику обробника**

При виклику обробника допустимим є передача адреси елемента, на якому відбулася подія або якоїсь його властивості. Для цього як параметр у функцію обробник передається **this,** який містить адресу поточного елемента або, відповідно, якоїсь його властивості this. властивість . В середині обробника this містить адресу елемента, для якого спрацював обробник (проводиться аналіз події). Також адресу елемента, для якого спрацював обробник можна одержати через об’єкт події event.currentTarget.

|  |  |
| --- | --- |
| Приклад | ----- Зробити напис на кнопці, на яку натиснуто, червоним-----  ------------- Елемент в HTML -------------  <input type="button" id="myButton" value="press" onclick="myHandler(this)"/>  ------------- Встановлення обробника в JavaScript --------------  <script>  function myHandler( btn )  {  //через btn одержуємо адресу кнопки, на яку натиснуто  btn.style.color ="red";  }  </script> |

=====================================================================================

**Об'єкт Event**

Об'єкт Event зберігає інформацію про подію і об’єкт який згенерував подію. Об’єкт Event передається у функцію обробник першим параметром

|  |  |
| --- | --- |
| Приклад | ----- Зробити напис на кнопці, на яку натиснуто, червоним-----  ------------- Елемент в HTML -------------  <input type="button" id="myButton" value=”Press” />  ------------- Встановлення обробника в JavaScript --------------  <script>  function myHandler( event )  {  var btn = event.target; //одержуємо адресу кнопки, на як натиснуто  btn.style.color ="red";  }  myButton.onclick=myHandler; // myHandler – функція обробник  </script> |

**type**    - тип події (наприклад, "click")

**target** – об'єкт, якому адресована подія (який згенерував подію)

**layerX, layerY** – горизонтальна і вертикальна позиції курсора миші

**which** – номер натиснутої кнопки миші.

**modifiers**– чи натиснута клавіша-модифікатор (Alt, Ctrl, Shift). Бітова маска, одна з констант ALT\_MASK, CONTROL\_MASK, SHIFT\_MASK, and META\_MASK

**data** – масив з рядків, який містить URL об'єктів, які були перетягнуті та впущені у вікно.

**pageX, pageY**– горизонтальна та вертикальна позиції курсора відносно веб-сторінки

**screenX, screenY**– горизонтальна та вертикальна позиції курсора на екрані

**width, height**– ширина та висота вікна після зміни розмірів.

srcElement - повертає посилання на елемент (тег) web-сторінки, викликав настання події;

fromElement - повертає посилання на елемент Web-сторінки, з якого перемістився курсор мишки при настанні події Mouseover чи Mouseout;

toElement - повертає посилання на елемент Web-сторінки, на який розміщено курсор мишки;

type - повертає назву події;

returnValue - вказує, чи виконуватиметься дія замовчуванням (якщо так буде, якщо false - немає);

propertyValue - повертає назву атрибут тега або стилю властивості елемента Web-сторінки, значення якого змінилося;

button - повертає номер натиснутою кнопкою мишки:

0 - нічого не було натиснуто;

1 - ліва кнопка натиснута;

2 - права кнопка натиснута;

3 - натиснуто одночасно ліва і права кнопки;

4 - середня кнопка натиснута;

5 - натиснуто одночасно ліва і середня кнопки;

6 - натиснуто одночасно права і середня кнопки;

7 - натиснуто одночасно всі кнопки.

--------------------------------------------------------------

Якщо потрібно передавати подія батьківського елементу, це можна заборонити, задавши властивості event.cancelBubble значення true.

// event.cancelBubble=true;

**Приклад. Малювання у компоненті canvas за допомогою миші.**

<canvas id="ccc" style="border:solid 2px"></canvas> //елемент, у якому здійснюємо графічні побудови

<script>

var b = false; //Змінна, яка містить true/false (прапорець – малювати/не\_малювати).

var x;

var y;

ccc.onmousedown = function (e) {

b = true; //Увімкнути режим малювання

x = e.clientX; //Отримати клієнтські координати миші

y = e.clientY;

}

ccc.onmouseup = function (e) {

b = false; //Відімкнути режим малювання

}

ccc.onmousemove = function (e) {

if (b) {

//var pos = document.getElementById("ccc").getBoundingClientRect();

var pos = this.getBoundingClientRect(); //Одержуємо інформацію про положення елемента

var cx = document.querySelector("canvas").getContext("2d");

cx.beginPath();

cx.moveTo(x - pos.left, y - pos.top); //Малюємо лінію від попередньої точки до поточної

x = e.clientX;

y = e.clientY;

cx.lineTo(x - pos.left, y - pos.top);

cx.stroke();

}

}

================================================================

**Приклад. Написати калькулятор**

### Захоплення подій

Наприклад, вам може знадобитися, щоб, об'єкт window обробляв всі події MouseDown, CLICK, MOUSEUP незалежно від місця їх виникнення.

Для цього існують методи

**captureEvents** – захоплює події вказаного типу.

**releaseEvents** – ігнорує події вказаних типів.

**routeEvent** – перенаправляє захоплену подію вказаному об'єкту.

**handleEvent** – обробляє захоплену подію.

window.captureEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP)

**Обробник події**

function clickHandler(e)

{

      var retval = routeEvent(e);

      if (retval == false) return false;

     // тут – обробка події

}

**Черга подій**

У процесі роботи браузера усі події, які генеруються як результат діяльності користувача, а також події, які згенеровані програмно потрапляють у чергу подій, з якої поступово ці події вибираються і опрацьовуються. Слід зазначити, що деякі дії користувача призводять до генерування одразу декількох подій. Наприклад, натискання на кнопку миші призводить до генерування подій : mousedown, mouseup, click.

**Підйом**

Звичайною ситуацією є випадок, коли один елемент знаходиться всередині іншого. При настанні події для деякого внутрішнього елемента відбувається послідовний виклик обробників даної події у напрямку від внутрішнього (для якого настала подія) до усіх зовнішніх, в середині яких він знаходиться.

Приклад. Розглянемо випадок, коли має 3 вкладних один в один елементи FORM > DIV > P, з обробниками подій на кожному з них:

<style>

body \* {

margin: 10px;

border: 1px solid blue;

}

</style>

<form onclick="alert('form')">FORM

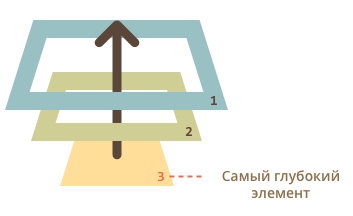
<div onclick="alert('div')">DIV

<p onclick="alert('p')">P</p>

</div>

</form>

У випадку настання події для внутрішнього елемента <p> поступово будуть викликані обробники подій спочатку для елементи який його містить, тобто <div>, а потім для елемента, який містить div, тобто form.



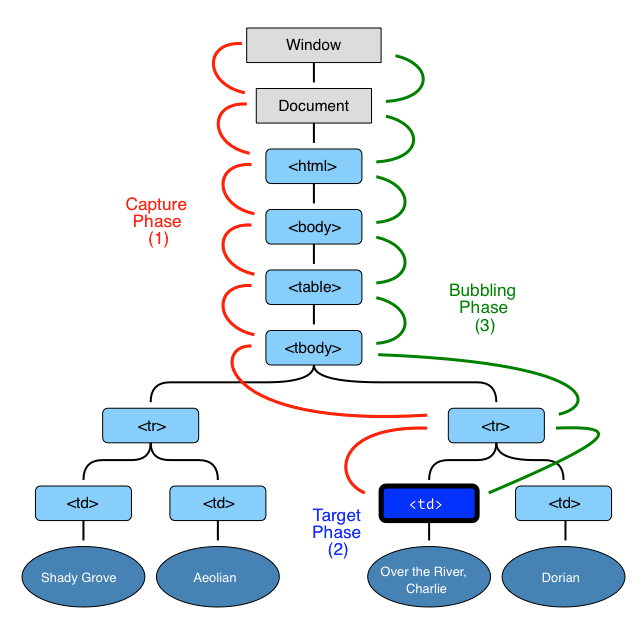
У випадку, якщо необхідно зупинити підйом (щоб не викликалися обробники елементів, які містять даний) можна скористатися методом об’єкта події event.stopPropagation() , (або з використанням властивосі **event.cancelBubble=true**). Кросбраузено:

event.stopPropagation ? event.stopPropagation() : (event.cancelBubble=true);

**Спуск**

У сучасному стандарті окрім підняття можна проаналізувати і етап спуску. Тому аналіз події складається з 3-х етапів.

* спуск (capturing stage);
* досягнення цільового елемента (target stage);
* підняття (bubbling stage).



У випадку, якщо необхідно проаналізувати подію на етапі спуску необхідно призначати функцію обробник за допомогою методу addEventListener з останнім параметром true, якщо на етапі підйому – з останнім параметром false.

Приклад.

<div id="divv" >

<input type="button" id="btn" name="name" value="" />

</div>

<script>

function f\_before(e) {

alert("before" + this.tagName + " target=" + e.target.tagName);

}

function f\_after(e) {

alert("after" + this.tagName + " target" + e.target.tagName);

}

divv.addEventListener("click", f\_before, true);

divv.addEventListener("click", f\_after, false);

</script>

Визначити етап, на якому аналізується подія, можна з використанням властивості

event.eventPhase – (спуск =1, підняття= 3).

Слід відмітити, що є події, які можна проаналізувати на етапі спуску, а на етапі підйому не можна.

Наприклад, такою подією є подія отримання фокусу на елементі [onfocus](https://learn.javascript.ru/focus-blur).

**Делегування обробки події**

Той факт, що при генеруванні події відбувається її підйом по ієрархії вкладеності (від внутрішнього до зовнішнього) часто використовують для реалізації делегування подій. Це означає, що не потрібно призначати для кожного внутрішнього елемента однотипний обробник однієї і тієї ж події. Це можна зробити для елемента, у якому вони знаходяться. Класичним прикладом може слугувати випадок, коли необхідно зробити аналіз події на клітинок таблиці (елементів «td», які знаходяться в середині елемента «table»). Зауважимо, що в середині клітинки можуть знаходитися інші елементи, і тоді визначення клітинки, у якій відбулася подія (наприклад, click), може бути здійснено або шляхом переходу від нащадка до предка (використовуючи властивість елемента parentElement), або з використанням методу елемента closest (визначення найближчого елемента вверх по ієрархії, які відповідає заданому селектору).

**Приклад**. Створити декілька кнопок, які знаходяться у контейнері div. Призначити метод обробки події click, при настанні якої напис з кнопки потрібно вставити у елемент вводу.



<div id="d1">

<input type="button" value="111" />

<input type="button" value="222" />

<input type="button" value="333" />

<input type="button" value="444" />

<input type="button" value="555" />

<input type="button" value="666" />

</div>

<input type="text" id="txt" value="" />

<script>

function f1(e) {

if (e.target.tagName=="DIV") { //Якщо цільовий елемент не кнопка, а div

txt.value = "div";

}

else

txt.value = e.target.value;

}

d1.onclick = f1; //Обробник призначаємо не кожній кнопці, а на контейнер, який їх містить

</script>

Приклад. Створити таблицю. При натисканні на клітинку вміст змінює на червоний колір.

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<table id="myTable" border="2px">

<tr>

<td>12</td>

<td><strong>55</strong> </td>

<td>12</td>

</tr>

<tr>

<td>12</td>

<td><strong>55</strong> </td>

<td>12</td>

</tr>

<tr>

<td>12</td>

<td><em><strong>55</strong></em> </td>

<td>12</td>

</tr>

</table>

<a href='file.html' onclick='return false;'>link</a>

<div onclick="alert('hello')">

<input type="button" id="myButton" value="press" />

</div>

<script>

function myFunc(event) {

var td = event.target.closest("td").style.color= "red";

// event.target.closest("td") – визначення найближчого до поточного елемента "td"

}

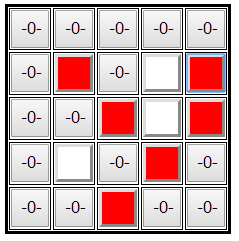
myTable.addEventListener("click", myFunc);

</script>

</body>

</html>

**Приклад**. Створити клас з грою «Міни». Випадковим чином у двовимірному масиві розміщують кнопки, для кожної з яких встановлюється атрибут, що містить 0-немає міни, 1- є міна.



<div id="forGame" >

</div>

<script>

class Mine {

constructor(rows, cols,idForGame)

{

this.rows = rows;

this.cols = cols;

this.idForGame = idForGame;

var \_table = this.createTable();

Object.defineProperty(this, "gameField", { value: \_table });

}

createTable()

{

var table = [];

for (var i = 0; i < this.rows; i++) {

var row = [];

for (var j = 0; j < this.cols; j++) {

row.push(Math.floor(Math.random() \* 2));

}

table.push(row);

}

return table;

}

onClick(e)

{

if (e.target.getAttribute("isMine")=="1") {

e.target.style.backgroundColor = "red";

}

else {

e.target.style.backgroundColor = "white";

}

e.target.innerText = " ";

}

render()

{

var ddiv = document.getElementById(this.idForGame);

var table = document.createElement("table");

table.style.border = "solid 2px";

for (var i = 0; i < this.rows; i++) {

var tr = document.createElement("tr");

for (var j = 0; j < this.cols; j++) {

var td = document.createElement("td");

td.style.border = "solid 1px";

var btn = document.createElement("button");

btn.style.width = "30px";

btn.style.height = "30px";

btn.innerText = "-0-";

//btn.onclick = this.onClick;

btn.setAttribute("isMine", this.gameField[i][j]);

td.appendChild(btn);

tr.appendChild(td);

}

table.appendChild(tr);

}

table.onclick = this.onClick; //Делегування, призначаємо обробник не на кожну кнопку а на таблицю

ddiv.appendChild(table);

}

}

//-------------------------------------

var game = new Mine(5, 5, "forGame"); // Створення об’єкта гри

game.render(); // Генерування розмітки

</script>

=======================================

**Генерування подій**

Для генерування події спочатку треба створити об’єкт класу Event і ініціювати подію з використанням методу elem.dispatchEvent(event)

**Приклад.**

|  |  |
| --- | --- |
| **Загальна форма** | var event = new Event(тип події[, опції]); |
| **Приклад** | var event = new Event("click"); |

Де:

* Тип события – может быть как своим, так и встроенным, к примеру "click".
* Флаги – объект вида { bubbles: true/false, cancelable: true/false }, где свойство bubblesуказывает, всплывает ли событие, а cancelable – можно ли отменить действие по умолчанию.

Флаги по умолчанию: {bubbles: false, cancelable: false}.

|  |  |
| --- | --- |
| **Приклад** | <button id="elem" onclick="alert('Клик');">Автоклик</button>  <script>  var event = new Event("click");  elem.dispatchEvent(event);  </script> |

### Добавление пользовательских данных – CustomEvent()[Раздел](https://developer.mozilla.org/ru/docs/Web/Guide/Events/%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5_%D0%B8_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9#%D0%94%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D1%85_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%E2%80%93_CustomEvent())

Чтобы добавить больше данных к объекту event, существует интерфейс [CustomEvent](https://developer.mozilla.org/ru/docs/Web/API/CustomEvent), и вы можете использовать свойство **detail**, чтобы передать собственные данные. Например, событие может быть создано так:

var event = new CustomEvent('build', { 'detail': elem.dataset.time });

Это позволит вам получить доступ к дополнительным данным в обработчике:

function eventHandler(e) {

log('The time is: ' + e.detail)

Задача 1. Розробити клас «Морський бій». Користувач має 10 снарядів. На полі 4\*4 розміщується випадковим чином 3 кораблі. Вивести за результатами гри відповідне повідомлення.

Задача 2. Гра «Відгадай слово». У конструктор передається слово, яке необхідно вгадати. На екрані знаходиться контейнер з елементами div для кожної букви, елемент вводу, у якому користувач вводить букву і натискає на кнопку. Після натиснення на кнопку необхідно відобразити букви у відповідних позиціях, якщо вони є, або повідомити користувача, що таких букв немає.

Задача 3. Гра «15». Початкова комбінація визначається випадковим чином.



<https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%B0_%D0%B2_15>

000000000000000000000000000000000000000000000000000000000000000000

# Координаты в окне

Для того, чтобы поместить один элемент рядом с другим на странице, а также двигать его произвольным образом, к примеру, рядом с указателем мыши – используются координаты.

Координатная система относительно окна браузера начинается в левом-верхнем углу текущей видимой области окна.

Мы будем называть координаты в ней clientX/clientY.

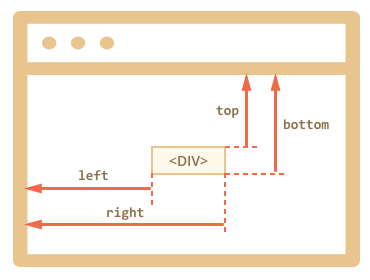
## [getBoundingClientRect()](https://learn.javascript.ru/coordinates" \l "getboundingclientrect)

Метод elem.getBoundingClientRect() возвращает координаты элемента, под которыми понимаются размеры «воображаемого прямоугольника», который охватывает весь элемент.

Координаты возвращаются в виде объекта со свойствами:

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

Например:



**Координаты относительно окна не учитывают прокрутку, они высчитываются от границ текущей видимой области.**

Иначе говоря, если страницу прокрутить, то элемент поднимется выше или опустится ниже – его координаты относительно окна изменятся.

Например, кликните на кнопку, чтобы увидеть её координаты:

Если вы прокрутите эту страницу, то положение кнопки в окне изменится, и её координаты, соответственно, тоже.

* Координаты могут быть дробными – это нормально, так как они возвращаются из внутренних структур браузера.
* Координаты могут быть и отрицательными, например если прокрутить страницу так, что верх элемента будет выходить за верхнуюю границу окна, то его top-координата будет меньше нуля.
* Некоторые современные браузеры также добавляют к результату getBoundingClientRect свойства для ширины и высоты: width/height, но их можно получить и простым вычитанием: height = bottom - top, width = right - left.

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

Если рассмотреть позиционирование элементов при помощи CSS-свойства position, то там тоже указываются left, right, top, bottom.

Однако, по CSS свойство right задаёт расстояние от правой границы, а bottom – от нижней.

Если вы взглянете на иллюстрацию выше, то увидите, что в JavaScript это не так. Все координаты отсчитываются слева/сверху, в том числе и эти.

**Метод elem.getBoundingClientRect() изнутри**

Браузер отображает любое содержимое, используя прямоугольники.

В случае с блочным элементом, таким как DIV, элемент сам по себе образует прямоугольник. Но если элемент строчный и содержит в себе длинный текст, то каждая строка будет отдельным прямоугольником, с одинаковой высотой но разной длиной (у каждой строки – своя длина).

Более подробно это описано в: [спецификации](http://www.w3.org/TR/CSS21/visuren.html#anonymous-block-level).

Если обобщить, содержимое элемента может отображаться в одном прямоугольнике или в нескольких.

Все эти прямоугольники можно получить с помощью [elem.getClientRects()](https://developer.mozilla.org/en/DOM/element.getClientRects). А метод [elem.getBoundingClientRect()](https://developer.mozilla.org/en/DOM/element.getBoundingClientRect) возвращает один охватывающий прямоугольник для всех getClientRects().

## [elementFromPoint(x, y)](https://learn.javascript.ru/coordinates" \l "elementFromPoint)

Возвращает элемент, который находится на координатах (x, y) относительно окна.

Синтаксис:

var elem = document.elementFromPoint(x, y);

Например, код ниже выделяет и выводит тег у элемента, который сейчас в середине окна:

var centerX = document.documentElement.clientWidth / 2;

var centerY = document.documentElement.clientHeight / 2;

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

elem.style.background = "red";

alert( elem.tagName );

elem.style.background = "";

Аналогично предыдущему методу, используются координаты относительно окна, так что, в зависимости от прокрутки страницы, в центре может быть разный элемент.

**Для координат вне окна elementFromPoint возвращает null**

Метод document.elementFromPoint(x,y) работает только если координаты (x,y) находятся в пределах окна.

Если одна из них отрицательна или больше чем ширина/высота окна – он возвращает null.

В большинстве случаев использования это не является проблемой, но нужно обязательно иметь такую возможность в виду.

Типичная ошибка, которая может возникнуть, если не проверять наличие elem:

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

// если координаты вне окна, то elem = null

elem.style.background = ''; // ошибка!

## [Координаты для position:fixed](https://learn.javascript.ru/coordinates" \l "%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B-%D0%B4%D0%BB%D1%8F-position-fixed)

Координаты обычно требуются не просто так, а, например, чтобы переместить элемент на них.

В CSS для позиционирования элемента относительно окна используется свойство position:fixed. Как правило, вместе с ним идут и координаты, например left/top.

Например, функция createMessageUnder из кода ниже покажет сообщение под элементом elem:

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

function createMessageUnder(elem, text) {

// получить координаты

var coords = elem.getBoundingClientRect();

// создать элемент для сообщения

var message = document.createElement('div');

// стиль лучше задавать классом

message.style.cssText = "position:fixed; color: red";

// к координатам обязательно добавляем "px"!

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

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

message.innerHTML = text;

return message;

}

// Использование

// добавить на 5 сек в документ

var message = createMessageUnder(elem, 'Привет, мир!');

document.body.appendChild(message);

setTimeout(function() {

document.body.removeChild(message);

}, 5000);

Нажмите на кнопку, чтобы запустить его:

кнопка с id=«coords-show-mark»

Этот код можно модифицировать, чтобы показывать сообщение слева, справа, сверху, делать это вместе с CSS-анимацией и так далее. Для этого нужно всего лишь понимать, как получить координаты.

**Заметим, однако, важную деталь: при прокрутке страницы сообщение будет визуально отдаляться от кнопки.**

Причина очевидна, ведь оно использует position: fixed, так что при прокрутке остаётся на месте, а страница скроллируется.

Как сделать, чтобы сообщение было именно на конкретном месте документа, а не окна, мы рассмотрим в следующей главе.

111111111111111111111111111111111111111111111111111111111111111111111

# Координаты в документе

Система координат относительно страницы или, иначе говоря, относительно документа, начинается в левом-верхнем углу, но не окна, а именно страницы.

И координаты в ней означают позицию по отношению не к окну браузера, а к документу в целом.

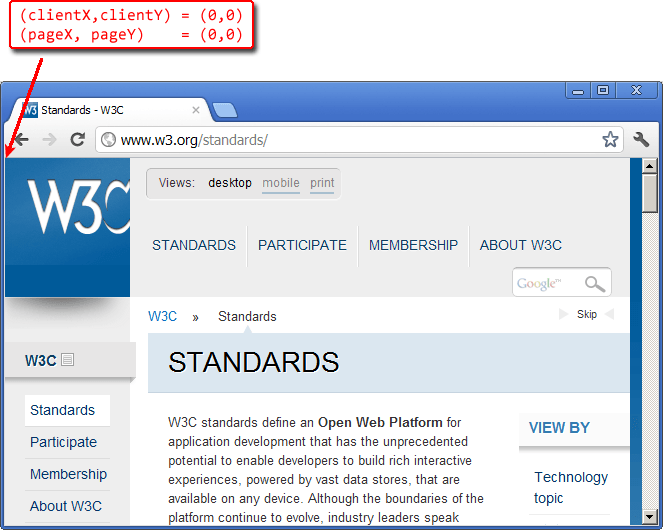
Если провести аналогию с CSS, то координаты относительно окна – это position:fixed, а относительно документа – position:absolute (при позиционировании вне других элементов, естественно).

Мы будем называть координаты в ней pageX/pageY.

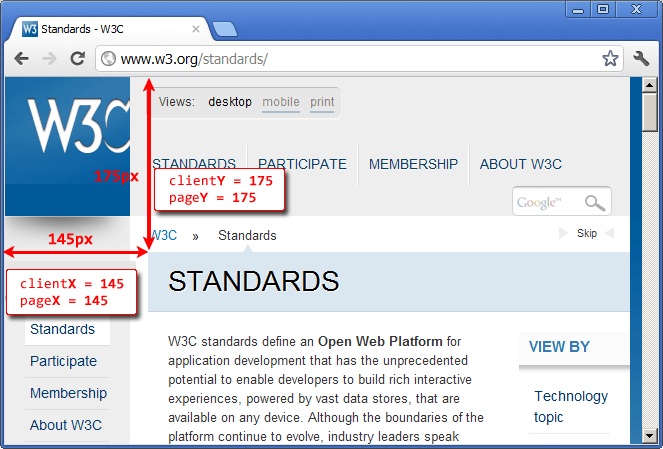
Они нужны в первую очередь для того, чтобы показывать элемент в определённом месте страницы, а не окна.

## [Сравнение систем координат](https://learn.javascript.ru/coordinates-document" \l "%D1%81%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC-%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82)

Когда страница не прокручена, точки начала координат относительно окна (clientX,clientY) и документа (pageX,pageY) совпадают:



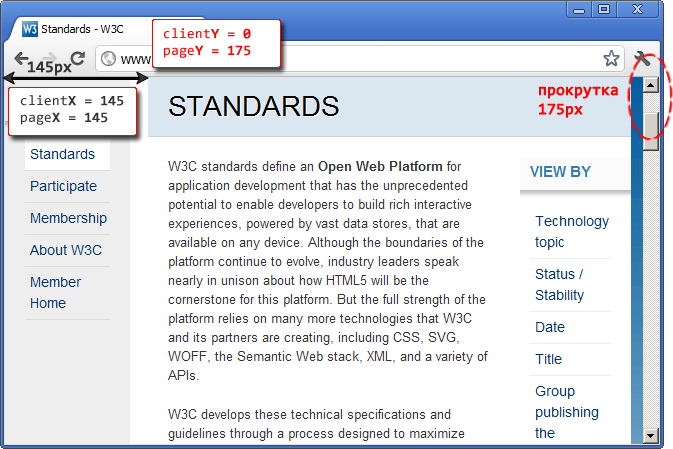
Например, координаты элемента с надписью «STANDARDS» равны расстоянию от верхней/левой границы окна:



**Прокрутим страницу, чтобы элемент был на самом верху:**

Посмотрите на рисунок ниже, на нём – та же страница, только прокрученная, и тот же элемент «STANDARDS».

* Координата clientY изменилась. Она была 175, а стала 0, так как элемент находится вверху окна.
* Координата pageY осталась такой же, так как отсчитывается от левого-верхнего угла документа.



Итак, координаты pageX/pageY не меняются при прокрутке, в отличие от clientX/clientY.

## [Получение координат](https://learn.javascript.ru/coordinates-document" \l "getCoords)

К сожалению, готовой функции для получения координат элемента относительно страницы нет. Но её можно легко написать самим.

Эти две системы координат жёстко связаны: pageY = clientY + текущая вертикальная прокрутка.

Наша функция getCoords(elem) будет брать результат elem.getBoundingClientRect() и прибавлять текущую прокрутку документа.

Результат getCoords: объект с координатами {left: .., top: ..}

function getCoords(elem) { // кроме IE8-

var box = elem.getBoundingClientRect();

return {

top: box.top + pageYOffset,

left: box.left + pageXOffset

};

}

Если нужно поддерживать более старые IE, то вот альтернативный, самый кросс-браузерный вариант:

function getCoords(elem) {

// (1)

var box = elem.getBoundingClientRect();

var body = document.body;

var docEl = document.documentElement;

// (2)

var scrollTop = window.pageYOffset || docEl.scrollTop || body.scrollTop;

var scrollLeft = window.pageXOffset || docEl.scrollLeft || body.scrollLeft;

// (3)

var clientTop = docEl.clientTop || body.clientTop || 0;

var clientLeft = docEl.clientLeft || body.clientLeft || 0;

// (4)

var top = box.top + scrollTop - clientTop;

var left = box.left + scrollLeft - clientLeft;

return {

top: top,

left: left

};

}

Разберем что и зачем, по шагам:

1. Получаем прямоугольник.
2. Считаем прокрутку страницы. Все браузеры, кроме IE8- поддерживают свойство pageXOffset/pageYOffset. В более старых IE, когда установлен DOCTYPE, прокрутку можно получить из documentElement, ну и наконец если DOCTYPE некорректен – использовать body.
3. В IE документ может быть смещен относительно левого верхнего угла. Получим это смещение.
4. Добавим прокрутку к координатам окна и вычтем смещение html/body, чтобы получить координаты нужного нам элемента.

### [Устаревший метод: offset\*](https://learn.javascript.ru/coordinates-document" \l "%D1%83%D1%81%D1%82%D0%B0%D1%80%D0%B5%D0%B2%D1%88%D0%B8%D0%B9-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4-offset)

Есть альтернативный способ нахождения координат – это пройти всю цепочку offsetParent от элемента вверх и сложить отступы offsetLeft/offsetTop.

Мы разбираем его здесь с учебной целью, так как он используется лишь в старых браузерах.

Вот функция, реализующая такой подход.

function getOffsetSum(elem) {

var top = 0,

left = 0;

while (elem) {

top = top + parseInt(elem.offsetTop);

left = left + parseInt(elem.offsetLeft);

elem = elem.offsetParent;

}

return {

top: top,

left: left

};

}

Казалось бы, код нормальный. И он как-то работает, но разные браузеры преподносят «сюрпризы», включая или выключая размер рамок и прокруток из offsetTop/Left, некорректно учитывая позиционирование. В итоге результат не всегда верен. Можно, конечно, разобрать эти проблемы и посчитать действительно аккуратно и правильно этим способом, но зачем? Ведь есть getBoundingClientRect.

Вы можете увидеть разницу между вычислением координат через offset\* и getBoundingClientRectна примере.

В прямоугольнике ниже есть 3 вложенных DIV. Все они имеют border, кое-кто из них имеет position/margin/padding.

Кликните по внутреннему (жёлтому) элементу, чтобы увидеть результаты обоих методов: getOffsetSum и getCoords, а также реальные координаты курсора – event.pageX/pageY (мы обсудим их позже в статье [Мышь: IE8-, исправление события](https://learn.javascript.ru/fixevent)).

Кликните, чтобы получить координаты getOffsetSum и getCoords

**getOffsetSum**:значение getOffsetSum()

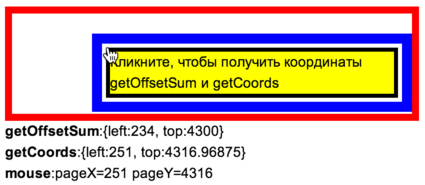
**getCoords**:значение getCoords()

**mouse**:координаты мыши

**При клике на любом месте желтого блока вы легко увидите разницу между getOffsetSum(elem) и getCoords(elem).**

Для того, чтобы узнать, какой же результат верный, кликните в левом-верхнем углу жёлтого блока, причём обратите внимание – кликать нужно не на жёлтом, а на чёрном, это рамка, она тоже входит в элемент. Будут видны точные координаты мыши, так что вы можете сравнить их с getOffsetSum/getCoords.

Пример клика в правильном месте (обратите внимание на разницу координат):

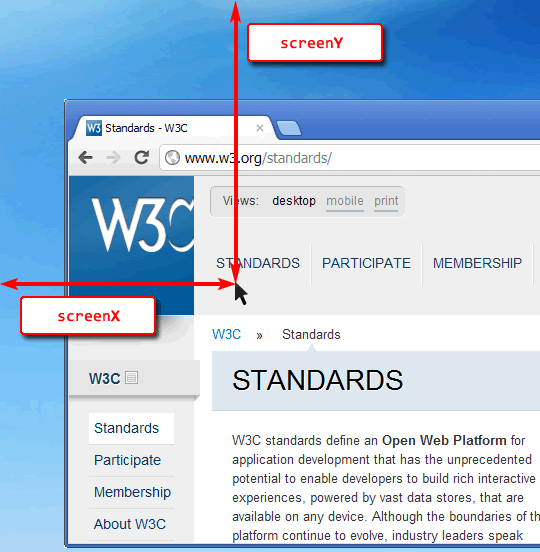


Именно getCoords всегда возвращает верное значение.

## [Координаты на экране screenX/screenY](https://learn.javascript.ru/coordinates-document" \l "%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D1%8B-%D0%BD%D0%B0-%D1%8D%D0%BA%D1%80%D0%B0%D0%BD%D0%B5-screenx-screeny)

Есть ещё одна система координат, которая используется очень редко, но для полноты картины необходимо её упомянуть.

Координаты относительно экрана screenX/screenY отсчитываются от его левого-верхнего угла. Имеется в виду именно весь экран, а не окно браузера.



Такие координаты могут быть полезны, например, при работе с мобильными устройствами или для открытия нового окна посередине экрана вызовом [window.open](https://developer.mozilla.org/en-US/docs/DOM/window.open).

* Размеры экрана хранятся в глобальной переменной [screen](https://developer.mozilla.org/en/DOM/window.screen):
* // общая ширина/высота
* alert( screen.width + ' x ' + screen.height );
* // доступная ширина/высота (за вычетом таскбара и т.п.)
* alert( screen.availWidth + ' x ' + screen.availHeight );

// есть и ряд других свойств screen (см. документацию)

* Координаты левого-верхнего угла браузера на экране хранятся в window.screenX, window.screenY(не поддерживаются IE8-):

alert( "Браузер находится на " + window.screenX + "," + window.screenY );

Они могут быть и меньше нуля, если окно частично вне экрана.

Заметим, что общую информацию об экране и браузере получить можно, а вот координаты конкретного элемента на экране – нельзя, нет аналога getBoundingClientRect или иного метода для этого.

## [Итого](https://learn.javascript.ru/coordinates-document" \l "%D0%B8%D1%82%D0%BE%D0%B3%D0%BE)

У любой точки в браузере есть координаты:

1. Относительно окна window – elem.getBoundingClientRect().
2. Относительно документа document – добавляем прокрутку, во всех фреймворках есть готовая функция.
3. Относительно экрана screen – можно узнать координаты браузера, но не элемента.

Иногда в старом коде можно встретить использование offsetTop/Left для подсчёта координат. Это очень старый и неправильный способ, не стоит его использовать.

Координаты будут нужны нам далее, при работе с событиями мыши (координаты клика) и элементами (перемещение).