**Web-дизайн**

Лабораторная работа №10

**Работа с событиями на веб-странице в JavaScript**

**События**

Для взаимодействия с пользователем в JavaScript определен механизм **событий**. По сути Действия пользователя, которые можно отследить и обработать через JavaScript, называются **событиями**.

Самый простой способ задать реакцию элемента на определенное событие – указать ее с помощью атрибута для определенного тега. К примеру, событию «нажатие мышкой» соответствует атрибут ***onclick***.

Значением атрибута с событием служит по сути JS-код:

<p onclick="alert('!')">Кликабельный текст</p>

А сейчас по клику на элемент выполнится функция *func*:

<p onclick="func()">Кликабельный текст</p>

<script>

function func() {alert('!');}

</script>

Так как это JS-код, то можно выполнить не одну функцию, а несколько:

<p onclick="func1();func2();">Кликабельный текст</p>

<script>

function func1() {alert('1');}

function func2() {alert('2');}

</script>

**Работа с this**

Специальный объект **this** указывает на элемент в котором произошло событие. Это уже готовая ссылка на объект. В примере **this** передается параметром функции и попадает в elem:

<script>

function func(elem) {alert(elem.value);}

</script>

<input type="submit" onclick="func(this)" value="input1">

<input type="submit" onclick="func(this)" value="input2">

**Обработчики событий**

До этого были рассмотрены *встроенные обработчики*, которые определяются в коде элемента с помощью атрибутов (к примеру, *onclick*). Однако этот способ имеет ряд недостатков, так что рекомендуется устанавливать обработчики в JS-коде.

Простейший способ – использовать *свойства обработчиков*. В JS-коде у элементов DOM имеются свойства обработчиков, соответствующие атрибутам:

<div id="rect"></div>

<script>

function *handler*(*e*){

    alert(*e*.type);

}

document.getElementById("rect").**onclick** = *handler*;

</script>

**Слушатели событий**

Еще один способ установки обработчиков событий представляет использование *слушателей* и является более оптимальным.

Для работы со *слушателями* событий в JavaScript есть объект **EventTarget**, который определяет методы **addEventListener**() для добавления *слушателя* и **removeEventListener**() для удаления *слушателя*. Так как html-элементы DOM являются объектами **EventTarget**, то они также имеют эти методы. Фактически *слушатели* представляют те же функции обработчиков.

Метод **addEventListener()** принимает два параметра: название события без префикса *on* и функцию обработчика этого события. Например:

<div id="rect"></div>

<script>

function handler(e){

    alert(e.type + "!");

}

var rect = document.getElementById("rect");

rect.**addEventListener**("*click*", function (e) {

    alert(e.type);

});

rect.**addEventListener**("*click*", handler); // можно установить функцию

</script>

Преимуществом использования слушателей является и то, что мы можем установить для одного события несколько функций.

Удаление слушателя аналогично добавлению:

rect.**removeEventListener**("click", handler);

**Объект Event**

При обработке события объект **Event** автоматически передается в функцию обработчика в качестве параметра. С помощью его свойств можно получить информацию о событии:

* *bubbles*: возвращает *true*, если событие является восходящим.
* *target*: указывает на элемент, на котором было вызвано событие.
* *type*: указывает на имя события.
* и т.д.

<div id="rect"></div>

<script>

function handler(**event**){

    console.log("Тип события: " + **event**.type + " " + **event**.target);

}

var rect = document.getElementById("rect");

rect.addEventListener("click", handler);

</script>

**Остановка выполнения обработчика события по умолчанию**

С помощью метода **preventDefault**() объекта **Event** можно остановить выполнение события для обработчика по умолчанию. В ряде случаев этот метод не играет большой роли. Однако в некоторых ситуаций он может быть полезен.

<a href="http://google.com" id="link">Поиск</a>

<script>

function linkHandler(e){

    var date = new Date();

    var hour = date.getHours();

    if(hour>12){ // остановит переход если больше 12 часов

        e.**preventDefault**();

        document.write("После 12 переход запрещен");

    }

}

var link = document.getElementById("link");

link.addEventListener("click", linkHandler);

</script>

**Распространение событий**

Когда нажимаем на элемент веб-станицы, генерируется событие нажатия. Это событие может распространяться от элемента к элементу. Например, если нажимаем на блок <div>, также нажимаем и на элемент <body>, где <div> находится. То есть происходит распространение события.

Есть несколько форм распространения событий:

* *Восходящие*: вверх по дереву DOM от дочерних узлов к родительским.
* *Нисходящие*: вниз по дереву DOM от родительских узлов к дочерним, пока не достигнет элемента, на котором это событие и возникло.

**Восходящие события (bubbling event)**

<div id="*outRect*">

    <div id="*inRect*"></div>

</div>

<script>

var *inRect* = document.getElementById("*inRect*");

*redRect*.addEventListener("click", function(){

    alert("Событие на inRect");

});

var *outRect* = document.getElementById("*outRect*");

*outRect*.addEventListener("click", function(){

    alert("Событие на outRect");

});

document.body.addEventListener("click", function(){

    alert("Событие на body");

});

</script>

Если нажмем на вложенный *div*, то событие пойдет к родительскому элементу *div* и далее к элементу *body*. Подобное поведение часто является нежелательным. В этом случае можно остановить распространение (*всплытие*) события с помощью метода **stopPropagation**():

*inRect*.addEventListener("click", function(e){

    alert("Событие на inRect");

    e.**stopPropagation**();

});

В результате нажатия событие будет обработано только обработчиком для *inRect*.

**Нисходящие события**

События также могут быть **нисходящими** (*capturing*). Для их использования в метод *addEventListener*() в качестве третьего необязательного параметра передается логическое значение *true* или *false*, которое указывает, будет ли событие нисходящим. По умолчанию событие восходящее. Если в предыдущем примере указать у слушателей *true*, то события будут распространяться в обратном порядке.

**События мыши**

Одну из наиболее часто используемых событий составляют события мыши:

* *click*: возникает при нажатии указателем мыши на элемент.
* *mousedown*: возникает при нахождении указателя мыши на элементе, когда кнопка мыши находится в нажатом состоянии.
* *mouseup*: возникает при нахождении указателя мыши на элементе во время отпускания кнопки мыши.
* *mouseover*: возникает при вхождении указателя мыши в границы элемента.
* *mousemove*: возникает при прохождении указателя мыши над элементом.
* *mouseout*: возникает, когда указатель мыши выходит за пределы элемента.

Например, обработаем события *mouseup* и *mousedown*:

<style>

.rectup{

width:100px;

height:100px;

margin:0px;

background-color:blue;

}

.rectdown{

width:70px;

height:70px;

margin:15px;

background-color:red;

}

</style>

<div class="rectup"></div>

<script>

function setColor(e){

if(e.type==="mousedown")

e.target.className="rectdown";

else if(e.type==="mouseup")

e.target.className="rectup";

}

var r = document.getElementsByClassName("rectup")[0];

r.addEventListener("mousedown", setColor);

r.addEventListener("mouseup", setColor);

</script>

Объект **Event** является общим для всех событий. Однако для разных типов событий существуют также свои объекты событий, которые добавляют ряд своих свойств. Так, для работы с событиями указателя мыши определен объект **MouseEvent**, который добавляет следующие свойства:

* *altKey*: возвращает *true*, если была нажат *Alt* во время генерации события.
* *shiftKey*: возвращает *true*, если была нажат *Shift* во время генерации события.
* *ctrlKey*: возвращает *true*, если была нажат *Ctrl* во время генерации события.
* *button*: указывает, какая кнопка мыши была нажата.
* *clientX*: определяет координату Х окна браузера, на которой находился указатель мыши во время генерации события
* *clientY*: определяет координату Y окна браузера, на которой находился указатель мыши во время генерации события.
* и т.д.

Определим координаты курсора на элементе:

<div class="rectup"></div>

<p id="p1"></p>

<script>

function handleClick(e){

p.innerText = "X: " + e.clientX + " Y: " + e.clientY;

}

var r = document.getElementsByClassName("rectup")[0];

var p = document.getElementById("p1");

r.addEventListener("mousemove", handleClick);

</script>

**События клавиатуры**

Другим распространенным типом событий являются события клавиатуры.

* *keydown*: возникает при нажатии клавиши клавиатуры и длится, пока нажата клавиша.
* *keyup*: возникает при отпускании клавиши клавиатуры.
* *keypress*: возникает при нажатии клавиши клавиатуры, но после события *keydown* и до события *keyup*. Надо учитывать, что данное событие генерируется только для тех клавиш, которые формируют вывод в виде символов. Нажатия на остальные клавиши, например, на *Alt*, не учитываются.

Для работы с событиями клавиатуры определен объект **KeyboardEvent**, который добавляет к свойствам объекта *Event* ряд специфичных для клавиатуры свойств:

* *altKey*: возвращает *true*, если была нажата клавиша *Alt* во время генерации события.
* *key*: возвращает символ нажатой клавиши, например, при нажатии на клавишу «T» это свойство будет содержать «T».
* *code*: возвращает строковое представление нажатой клавиши физической клавиатуры QWERTY, например, при нажатии на клавишу «*T*» это свойство будет содержать «*KeyT*», а при нажатии на клавишу «*;*», то свойство возвратит «*Semicolon*».
* *ctrlKey*: возвращает *true*, если была нажат *Ctrl* во время генерации события.
* *shiftKey*: возвращает *true*, если была нажат *Shift* во время генерации события.
* и т.д.

Например, можно с помощью клавиш клавиатуры перемещать элемент на веб-странице:

<style>

body{ margin:0; }

#rect{

width:100px;

height:100px;

margin-left: 0px;

margin-top: 0px;

background-color:blue;

}

</style>

<div id="rect"></div>

<script>

function moveRect(e){

var r = document.getElementById("rect");

// получаем стиль для r в числовых значениях

var cs = window.getComputedStyle(r);

var left = parseInt(cs.marginLeft);

var top = parseInt(cs.marginTop);

switch(e.key){

case "ArrowLeft": // если нажата клавиша влево

r.style.marginLeft = left - 10 + "px";

break;

case "ArrowUp": // если нажата клавиша вверх

r.style.marginTop = top - 10 + "px";

break;

case "ArrowRight": // если нажата клавиша вправо

r.style.marginLeft = left + 10 + "px";

break;

case "ArrowDown": // если нажата клавиша вниз

r.style.marginTop = top + 10 + "px";

break;

}

}

addEventListener("keydown", moveRect);

</script>

**Работа с формами**

Один из способов взаимодействия с пользователями представляют html-формы. В JavaScript форма представлена объектом **HtmlFormElemen**t.

<form name="search">

<input type="text" name="field1" value="1">

<input type="submit" name="send1" value="2">

</form>

<form name="setting">

<input type="text" name="field2" value="1">

<input type="submit" name="send2" value="2">

</form>

Первый способ обращения к форме заключается в прямом обращении по имени формы:

var searchForm = document.search; // форма с именем search

Второй способ состоит в обращении к коллекции форм (**document.forms**) и поиске в ней нужной формы:

var searchForm;

var forms = document.**forms**;

for (var i = 0; i < forms; i++)

if(forms[i].name === "search")

  searchForm = forms[i];

}

Так как *document.forms* ассоциативный массив, к формам можно обращаться по имени вместо номера.

var searchForm = document.forms['search'];

**Обращение к свойствам форм**

У каждой формы есть свойство *elements*, которое содержит массив элементов (свойств) формы. С его помощью можно обратиться к определенному элементу формы по его номеру.

Элементы формы имеют атрибут **name**, по которым можно обращается к конкретным элементам. Другими важным свойствами является свойство **value** и **type.**

var elements = document.forms[1].**elements**;

for (var i = 0; i < elements.length; i++) {

var elName = elements[i].name; // получаем имя элемента формы

// получаем элемент по имени

alert(elements[elName].value); // получаем значение элемента формы

alert(elements[elName].type); // получаем тип элемента формы

}

Среди методов формы надо отметить метод **submit**(), который отправляет данные формы на сервер, и метод **reset**(), который очищает поля формы. Также есть методы **focus**() (устанавливает фокус на элемент) и **blur**() (убирает фокус с элемента).

**Кнопки**

Для отправки введенных данных на форме используются **кнопки**. Для создания кнопки используется элемент **button** илиэлемент **input**:

<button name="send">Отправить</button>

<input type="submit" name="send" value="Отправить">

При нажатии на любой из этих двух вариантов кнопки происходит отправка формы по адресу, который указан у формы в атрибуте *action*, либо по адресу текущей веб-страницы (работа с отправленными данными производится на сервере).

В коде javascript можно перехватить отправку, обрабатывая событие *click*:

<form name="search">

    <input type="text" name="key">Введите текст</input>

    <input type="submit" name="send" value="Отправить" />

</form>

<script>

function sendForm(e){

    var val = document.search.key.value; // получаем значение поля key

    if(val.length > 5){

        alert("Недопустимая длина строки");

        e.preventDefault(); // прерываем выполнение события

    }

    else alert("Отправка разрешена");

}

var sendButton = document.search.send;

sendButton.addEventListener("click", sendForm);

</script>

Помимо кнопок внутри формы могут располагать различные поля, такие как: текстовые поля (<input type="text">), флаги (<input type="checkbox">), переключатели (<input type="radio">), списки (<select>) и т.д. Эти элементы позволяют определить пользовательские данные, которые будет обрабатывать веб-приложение.

Знакомство с JS - <https://webref.ru/dev/learn-javascript>

Самоучитель JS - <https://learn.javascript.ru/>

Самоучитель JS - <https://metanit.com/web/javascript/>

Справочник JS - <https://javascript.ru/manual>

Справочник JS - <https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference>

**Общие задания**

Выполните следующие задания с использованием JS:

1. Создайте поле для ввода и кнопку. Сделайте так, чтобы при нажатии на кнопку данные, введенные пользователем, добавлялись в массив (изначально массив пустой). Отображайте этот массив на странице в виде html-списка. Поле после нажатия должно очищаться. Проводите проверку на пустую строку в поле для ввода.
2. Создайте 3 кнопки. При наведении на них их текст (value) должен меняться на «Наведен». При уходе курсора с них текст должен меняться на «Не наведен». При нажатии на кнопку цвет текста кнопки должен меняться на случайный. Сделайте это через функции на каждое событие, но общие для всех кнопок (с помощью *this*).
3. Дан *html*-список, в нем несколько элементовс текстом «Новый». Под спискомсделайте кнопку, по нажатию на которую в конец будет добавляться новый  элемент с текстом «Новый». Сделайте так, чтобы при нажатии на элемент списка (включая добавленные), текст менялся на «Старый». При повторном нажатии – на «Очень старый». При нажатии на «очень старый» элемент, тот должен удаляться из списка.
4. Создайте элемент *div*. Сделайте его круглым. Сделайте так, чтобы при нажатии пользователя на окно браузера, координаты нажатия запоминались. Элемент должен постепенно двигаться в эту точку координат. Постарайтесь сделать движение плавным (можно использовать *requestAnimationFrame()* и *transfotm: translate()* для перемещения). Если пользователь снова нажмет на экран браузера, то элемент должен сменить свое направление движения на новую точку.
5. Создать арифметический калькулятор с управлением с клавиатуры. Для отображения вводимого числа (а также результата) используйте что угодно (*div*, *p*, *span* и т.п.) **кроме** полей ввода (**не использовать** *input*). Определяйте, что нажал пользователь с помощью обработки событий на клавиатуре. Должна быть обработка для ввода чисел и арифметических операций (символы +, -, \*, /), а также для поучения результата (=).

**Индивидуальное задание**

Дополните сайт, разработанный по выбранной теме по индивидуальному заданию с прошлых лабораторных работ.

Под меню добавьте небольшую область, где с определенной периодичностью будет возникать краткое описание страниц сайта по порядку (вначале описание первой, затем второй, заменяя первое описание, и т.д.).

Отображайте текущее время в шапке страницы, которое обновляется каждую секунду.

Также добавьте функцию, которая выполниться, если долго не переключать страницу. Она должна добавлять в центр страницы блок с «рекламой». На этом блоке также должна быть кнопка «закрыть» (к примеру, меняет *display* блока на *none*). Для размещения и фиксирования блока можно использовать css-свойство *position* со значением *fixed* (чтобы даже при прокрутки страницы вниз блок все равно оставался на месте).

Разделите ссылки в меню на две группы. Ссылки изначально не должны быть видны. Они должны появляться в качестве выпадающего меню **при нажатии** на одну из групп (должны показываться соответствующие ссылки). Нажатая кнопка должна как-то выделяться (меняться оформление).

На страницу с таблицей добавьте поля ввода данных строки и кнопку «Добавить». При нажатии на кнопку должен создаваться объект по введенным данным в поля и добавляться в массив объектов и таблицу.

**Дополнительно:**

С помощью метатегов можно добавить на страницу информацию полезную для поисковых систем: краткое описание и ключевые слова на каждую страницу в <head>:

<meta name="keywords" content="важные, ключевые, слова">

<meta name="description" content="краткое описание">

Ключевые слова по идее должны помогать поисковым системам в обнаружении вашей страницы по целевым запросам пользователей. Краткое описание (или аннотация) страницы часто используется поисковиками при отображении результатов поиска.