**Лабораторная работа №10**

**Тема работы:** Разработка программ с обращением к объектам DOM

**Цель работы:** закрепить навыки по работе с элементами DOM.

**Теоретические сведения**

**Основы работы с DOM в JavaScript**

В JavaScript существует **несколько способов** работы с событиями. Мы начнем с самого простого из них.

**Основы работы с 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**, событию *'наведение мышкой'* - атрибут **onmouseover**, а событию *'уход курсора с элемента'* - атрибут **onmouseout**.

Значением атрибута с событием служит **JavaScript код**. В следующем примере *по нажатию мышкой на кнопку* выполнится функция **alert**:

<input type="submit" onclick="alert('!')">

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

<input type="submit" onclick="func()">

function func() {

alert('!');

}

Можно выполнить **не одну функцию**, а несколько:

<input type="submit" onclick="func1(); func2();">

function func1() {

alert('1');

}

function func2() {

alert('2');

}

Обратите внимание на то, что если внутри атрибута вам нужны двойные кавычки (например, для строки) и внешние кавычки атрибута тоже двойные - *onclick="alert("!")"* - такой код не будет работать.

С этим можно бороться несколькими способами: можно сменить внешние кавычки на одинарные *onclick='alert("!")'*, можно также заэкранировать внутренние кавычки обратным слешем *onclick="alert(\"!\")"* или же просто перенести JavaScript код из атрибута в функцию, а в атрибуте оставить только имя функции *onclick="func()"*.

То же самое будет, если вы **внешние** кавычки атрибута ставите одинарными и для строки тоже используете одинарные: *onclick='alert('!')'* - тут также все решается аналогичными способами.

**Таблица атрибутов для событий**

**События мыши**

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| onclick | Произведен клик кнопкой мыши на элементе управления. Событие возникает после того, как кнопка мыши была отпущена. |
| onmousedown | Нажата кнопка мыши. |
| onmouseup | Отжата кнопка мыши. |
| ondblclick | Двойной щелчок. |
| onmousemove | Указатель мыши движется внутри области отображения элемента. |
| onmouseout | Указатель мыши вышел из области отображения элемента. |
| onmouseover | Указатель мыши находится внутри области отображения элемента. |

**События клавиатуры**

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| onkeydown | Нажата кнопки на клавиатуре. |
| onkeypress | Кнопка на клавиатуре нажата и не отпускается дольше, чем интервал повторения. Длительность интервала повторения является системным параметром и зависит от настроек операционной системы пользователя. |
| onkeyup | Отпущена ранее нажатая кнопка. |

**События форм и элементов страницы**

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| onchange | Элемент теряет фокус ввода, а содержимое элемента изменилось за время, пока элемент был в фокусе. |
| onselect | Какая-то часть текста внутри элемента становится выделенной. |
| onsubmit | В форме нажата кнопка "Отправить", но отправка формы на сервер еще не производилась. |

**События форм и элементов страницы**

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| onblur | Элемент управления теряет фокус ввода, т.е. курсор переходит к другому элементу. |
| onfocus | Отображаемый элемент получил фокуса ввода. Для текстовых полей это событие означает, что курсор уже находится в данном элементе. |
| onload | Завершена загрузка страницы. |
| onunload | Производится выход из документа (закрытие или перенаправление страницы на другой адрес). |
| onscroll | Срабатывает при прокрутке элемента. |

**Работа с getElementById**

Сейчас мы с вами научимся получать элементы **HTML** страницы и проводить с ними различные **манипуляции** (мы сможем менять, к примеру, их текст и цвет и многие другие полезные вещи).

Пусть у нас на странице есть тег с атрибутом **id** в значении **test**. Запишем ссылку на этот тег в переменную **elem**. Для этого мы должны воспользоваться методом **getElementById**, который получает элемент по его **id**.

Эта запись произойдет **по клику** на кнопку, которой мы задали атрибут **onclick**. По нажатию на эту кнопку сработает функция **func**, которая найдет на **HTML** странице элемент с **id** равным **test** и запишет *ссылку на него* в переменную **elem**:

<input type="text" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

}

Теперь в переменной **elem** у нас лежит ссылка на элемент с атрибутом **id** в значении **test**. Сама переменная **elem** является *объектом*.

Этот объект и тег HTML страницы связаны друг с другом - мы можем поменять какие-либо свойства объекта **elem** и при этом увидим изменения на HTML странице, которые произойдут с полученным нами элементом.

Давайте посмотрим, как это происходит на практике.

**Основы работы с атрибутами HTML через JavaScript**

Сейчас мы будем считывать и изменять атрибуты тегов. Пусть у нас опять даны **инпут** с id равным test и **кнопка**, по клику на которую будет запускаться функция **func**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

Внутри функции **func** мы получим наш инпут *по его id* и запишем ссылку на него в переменную **elem**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

}

Давайте теперь выведем на экран содержимое атрибутов нашего инпута. Чтобы получить доступ, к примеру, к атрибуту value, следует написать следующее: **elem.value**, где elem - это переменная, в которую мы с помощью getElementById записали ссылку на наш элемент, а value - это атрибут тега, который нас интересует.

Мы можем вывести содержимое атрибута через alert таким образом - **alert(elem.value)** - или записать в какую-нибудь переменную. Давайте проделаем это:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

alert(elem.value); //выведет '!'

}

Мы можем таким же образом считывать значения и других атрибутов, например так - **elem.id** - мы считаем значение атрибута id, а так - **elem.type** - значение атрибута type. Смотрите пример:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

alert(elem.value); //выведет '!'

alert(elem.id); //выведет 'test'

alert(elem.type); //выведет 'text'

}

Можно не только считывать значения атрибутов, но и **изменять** их. Чтобы, к примеру, поменять значение атрибута **value**, нужно просто присвоить его конструкции **elem.value**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.value **=** 'www'; //присвоим новое значение атрибуту value

}

HTML код станет выглядеть так (значение атрибута value станет www):

<input type="text" value="www" id="test">

<input type="submit" onclick="func()">

Ну, а теперь самое сложное - можно **не вводить** переменную **elem**, а строить **цепочку** из точек таким образом:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

alert(document.getElementById('test').value); //выведет '!'

}

Таким же образом (цепочкой) можно производить и **перезапись атрибутов**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

document.getElementById('test').value **=** 'www';

}

Однако, в большинстве случаев введение переменной **удобнее**. Сравните два примера - сейчас я ввел переменную **elem** и могу считывать любое количество атрибутов, при этом **getElementById** вызывается *только один раз*:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.value **=** 'www';

elem.type **=** 'submit';

}

А сейчас я не ввожу новую переменную и поэтому мне приходится вызывать **getElementById** два раза:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

document.getElementById('test').value **=** 'www';

document.getElementById('test').type **=** 'submit';

}

На мой взгляд, этот код стал **сложнее**, хотя и занимает на одну строчку меньше. Кроме того, если я захочу сменить значение **id** с test на, к примеру, www, мне придется делать это во многих местах, что не очень удобно.

Есть и еще проблема - **нагрузка** на браузер. Поиск элементов по странице, который делает метод **getElementById**, является довольно **медленной** операцией (*и вообще любая работа с элементами страницы - это медленная операция* - запомните это).

В нашем случае, если мы каждый раз используем **getElementById**, то браузер каждый раз будет обрабатывать HTML страницу и искать элемент с заданным id **несколько** раз (не важно, что id одинаковые - браузер проделает все действия несколько раз), совершая бесполезные операции, которые могут замедлить работу браузера.

Если же мы используем переменную **elem** - никакого поиска по странице не происходит (элемент уже найден и ссылка на него лежит в переменной **elem**).

**Исключения: атрибуты class и for**

Вы уже научились работать с атрибутами через JavaScript и теперь пришло время рассказать вам о том, что **не все так просто** - при работе с атрибутами существует исключение - это атрибут **class**.

Это слово является **специальным** в JavaScript и поэтому мы не можем просто написать *elem.class*, чтобы считать значение атрибута **class**. Вместо этого следует писать *elem.className*.

В следующем примере на экран выводится значение атрибута **class**:

<input type="text" class="aaa bbb" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

alert(elem.className);

}

Кстати, есть и другие атрибуты, которые называются иначе, чем свойство. Например, атрибуту **for** (<label for="...">) соответствует свойство с названием **htmlFor**.

**Работа с this**

Сейчас мы с вами будем работать со специальным объектом **this**, который указывает на текущий элемент (элемент в котором произошло событие). Причем указывает так, будто этот элемент уже получен методом **getElementById**.

Давайте посмотрим как работать с **this** и в чем удобство такого подхода.

Пусть у нас есть задача по нажатию на инпут вывести на экран содержимое его value.

Пока вы умеете делать только такое решение:

<input type="submit" onclick="func()" id="test" value="!">

function func() {

var elem **=** document.getElementById('test');

alert(elem.value);

}

В принципе, это решение хорошее, но представим теперь, что у нас есть много инпутов и по нажатию на каждый нам нужно выводить его value.

В этом случае у нас получится что-то в таком роде:

<input type="submit" onclick="func1()" id="test1" value="input1">

<input type="submit" onclick="func2()" id="test2" value="input2">

<input type="submit" onclick="func3()" id="test3" value="input3">

function func1() {

var elem **=** document.getElementById('test1');

alert(elem.value);

}

function func2() {

var elem **=** document.getElementById('test2');

alert(elem.value);

}

function func3() {

var elem **=** document.getElementById('test3');

alert(elem.value);

}

Теперь ясно виден недостаток нашего подхода - для каждого инпута нам приходится создавать свою функцию обработки клика, причем делают эти функции практически одно и тоже.

Если у нас будет 10 инпутов - то придется сделать 10 функций, не удобно.

Давайте упростим нашу задачу: будем передавать параметром функции id текущего элемента. И вместо большого количества функций все сведется к одной функции:

<input type="submit" onclick="func('test1')" id="test1" value="input1">

<input type="submit" onclick="func('test2')" id="test2" value="input2">

<input type="submit" onclick="func('test3')" id="test3" value="input3">

function func(id) {

var elem **=** document.getElementById(id);

alert(elem.value);

}

Однако, такое решение все равно имеет недостаток - каждому элементу придется вводить разные id, что тоже несколько неудобно.

Итак, давайте наконец рассмотрим вариант решения задачи через **this**.

Сделаем так, что каждый инпут будет выводить свое содержимое по нажатию. Для этого параметром функции передадим объект **this**, вот так: **func(this)**.

Этот this - это уже готовая ссылка на объект. То есть если я кликаю на первый инпут - в this окажется ссылка на него, если на второй инпут - то на него, и так далее.

Наш this передается параметром функции и попадает в переменную **elem**. Этот elem ведет себя так, будто получен таким образом: *var elem = document.getElementById(...)*, но получать его таким образом не надо, там уже все готово и можно пользоваться. К примеру, **elem.value** указывает на value нашего инпута и так далее.

Итак, вот самое просто решение нашей задачи:

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

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

<input type="submit" onclick="func(this)" value="input3">

function func(elem) {

alert(elem.value);

}

**Основы работы с CSS**

В **JavaScript** работа с **CSS свойствами** происходит путем изменения значения [атрибута style](http://old.code.mu/attr/style.html) для элемента. К примеру, чтобы поменять **цвет** нужно построить следующую цепочку - *elem.style.color* - и присвоить ей нужное значение цвета:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.style.color **=** 'red';

}

Можно также и не вводить переменную **elem**, а построить *очень длинную цепочку*:

function func() {

document.getElementById('test').style.color **=** 'red';

}

Существует много **CSS** свойств, которые пишутся **через дефис**, например, font-size. В этом случае оно преобразуется в **fontSize**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.style.fontSize **=** '15px';

}

Работу с кроссбраузерными приставками рассмотрим на примере свойства -moz-box-sizing - оно преобразуется в **mozBoxSizing** (остальные свойства с приставками аналогично):

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.style.mozBoxSizing **=** 'border-box';

}

Обратите внимание на то, что свойство float является исключением (оно является специальным в JavaScript) и вместо него следует писать **cssFloat**:

<input type="text" value="!" id="test">

<input type="submit" onclick="func()">

function func() {

var elem **=** document.getElementById('test');

elem.style.cssFloat **=** 'left';

}

**Порядок выполнения работы**

**Практическая часть**

В папке «ЛР10Практика» запустить файл «index.html» и выполнить оттуда примеры и задачи.