**Введение**

Объектная модель документов (DOM) определена консорциумом World Wide Web Consortium в трех группах спецификаций (DOM Level 1, DOM Level 2 и DOM Level 3). DOM представляет HTML- или XML-документ в виде иерархического дерева элементов, каждый из которых обладает определенными свойствами и имеет определенные методы. Используя клиентские языки, такие как JavaScript, можно добавлять, изменять, удалять и назначать события элементам дерева, что позволяет создавать интерактивные, динамические Web-страницы.

**DOM-скриптинг**

В терминологии DOM объект Document является корнем дерева. В JavaScript это window.document, или просто Document (так как он присоединен к объекту Window). Это отправная точка для некоторых реализаций JavaScript. В листинге 1 приведен пример фрагмента HTML-кода.

**Листинг 1. HTML-код**

<body>

<p id="paragraph1">

<span>This is some text</span>

<a href="/index.html" title="Click here">Click here</a>

<p>

</body>

С точки зрения DOM тег p в этом примере представляется посредством DOM-интерфейса Element. Этот тег является родительским по отношению к тегам span и a. Теги span и a являются элементами одного уровня.

Предположим, что мы хотим получить атрибут href якоря из листинга 1. Простым способом получения доступа к элементу в DOM является использование метода getElementById. Следующий код, написанный на языке описания интерфейсов (Interface Definition Language, IDL), представляет собой часть описания интерфейса документа и содержит имя и список параметров методаgetElementById : Element getElementById (in DOMString elementId).

Интерфейс DOMString реализован в JavaScript посредством объекта String; таким образом, вышеуказанный метод принимает идентификатор элемента в виде параметра строкового типа. В нашем примере тег p является единственным элементом, содержащим атрибут id, поэтому его можно извлечь с помощью конструкции var paragraph = document.getElementById("paragraph1");.

Чтобы получить якорь, заключенный в тег p, можно использовать атрибут childNodes. Этот атрибут является частью интерфейсаNode и возвращает объект типа NodeList. Объект является array-like объектом в JavaScript. Array-like объекты не имеют методов, например, pop() или push(), но имеют свойство length. Объект, возвращаемый атрибутом childNodes, не различает узловые элементы (HTML-теги), текстовые элементы и элементы-комментарии. Для поиска только узловых элементов можно использовать атрибут children. Поскольку нам не нужны текстовые элементы или комментарии, для нашего случая будет лучше использовать именно этот атрибут, а не атрибут childNodes. В нашем примере якорь является вторым дочерним элементом параграфа, так что его можно получить с помощью следующей конструкции: var aElement = paragraph.children[1];.

Для получения значения атрибута href имеющегося у нас элемента можно использовать метод getAttribute, передав ему в качестве параметра имя атрибута (в нашем случае href). Часть IDL-описания, содержащая метод getAttribute, выглядит так:DOMString getAttribute (in DOMString name).

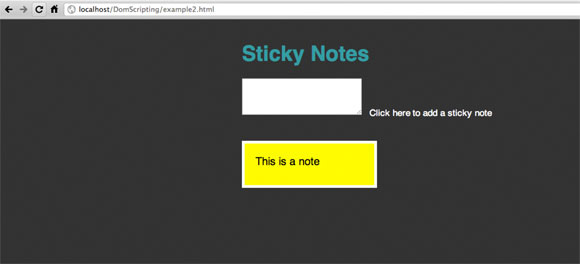
В нашем примере можно реализовать вышеупомянутый интерфейс следующим образом: var aHref = aElement.getAttribute("href"); // "index.html".

Так же как и в JavaScript, методы можно связывать в цепочку. Например, значение атрибута href тега a можно получить с помощью одной строки кода: var aHref = document.getElementById("paragraph1").children[1].getAttribute("href"); // index.html \*/.

**Углубляемся в DOM-скриптинг: тестовое приложение**

В этом разделе мы рассмотрим некоторые возможности DOM-скриптинга. Мы рассмотрим тестовое приложение Sticky Notes (заметки на липучках), которое представляет собой интерактивную Web-страницу, на которую пользователь может "приклеивать" заметки без ее перезагрузки. Эта Web-страница изображена на рисунке 1.

**Рисунок 1. Клиентский интерфейс приложения Sticky Notes**



HTML-код этой страницы представлен в листинге 2. Внутри тега head находятся ссылки на файлы CSS и JS. Внутри тега body можно увидеть структуру заметок, которые уже присутствуют на странице – это тег textarea и якорь, при срабатывании которого создается новая заметка.

**Листинг 2. HTML-код приложения**

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8">

<title>

Dom Scripting

</title>

<link rel="stylesheet" href="css/master.css" />

<script src="js/script.js"></script>

</head>

<body>

<div class="wrapper">

<h1> Sticky Notes </h1>

<div class="links">

<textarea id="contentArea" cols="10"> </textarea>

<a href="/random.html" class="add">Click here</a>

<span>to add a sticky note</span>

</div>

<div id="notes">

<div class="note">

<p>

This is a note

</p>

</div>

</div>

</div>

</body>

</html>

Давайте проанализируем JavaScript-код в файле script.js, который загружается при открытии страницы. Этот сценарий должен выполняться один раз при загрузке страницы или при создании структуры документа. Чтобы добиться этого, можно привязать функцию к оконному атрибуту onload, как показано в листинге 3.

**Листинг 3. Атрибут onload**

window.onload = init;

function init() {}

Атрибут onload связан с DOM-событием загрузки (load), что является типовым способом привязки события к функции прослушивателя событий на уровне DOM Level 0 (эта "спецификация" поддерживается всеми браузерами, но не является стандартом). Стандартная модель DOM Events Model определена внутри спецификаций DOM Level 2. В этой спецификации для регистрации обработчика событий в конечном элементе определен метод addEventListener (из интерфейса EventTarget). Этот метод описывается следующим кодом: object.addEventListener(eventType, eventHandler, useCapture);.

Здесь eventType – это регистрируемое событие объекта, eventHandler – функция для привязки к определенному событию,useCapture – необязательное булево значение, определяющее, на какой фазе события (всплывание или захват) будет вызвана функция. В следующем коде функция addEventListener используется для привязки метода load к окну:window.addEventListener("load", init, false);.

К сожалению, Internet Explorer версии ниже 9 не поддерживает вышеописанный метод W3C и имеет свою собственную реализацию:object.attachListener(eventType, eventHandler);.

Для параметра eventType необходимо указывать перед именем события префикс on. По умолчанию события в IE являются всплывающими, поэтому параметр useCapture отсутствует.

В листинге 4 приведен фрагмент кода из файла script.js, содержащий функцию addEvent, которая управляет привязкой событий во всех браузерах. Это метод глобального объекта под названием SA. Он работает в любом из рассмотренных выше вариантов.

**Листинг 4. Функция addEvent**

window.SA = {

addEvent : function(element, evType, fn, useCapture) {

if (element.addEventListener) {

element.addEventListener (evType, fn, useCapture);

return true;

} else if (element.attachEvent) {

var r = element.attachEvent('on' + evType, fn);

return r;

} else {

element['on' + evType] = fn;

}

}

}

Если вы используете функцию addEvent, то можете привязать функцию (давайте назовем ее SA.load) к событию load, как показано в листинге 5.

**Листинг 5. Привязка функции**

SA.addEvent(window, "load", SA.load, false);

SA = {

...

load : function() {

// init block

}

}

Функция SA.load из вышеприведенного примера срабатывает только тогда, когда загружены все ресурсы, поскольку она привязана к событию load. В общем случае перед выполнением функции, привязанной к событию загрузки (load), может пройти какое-то время, особенно если загружаемая страница содержит большое количество графических изображений. Хорошей практикой является привязывать функцию инициализации сценария к событию DOMContentLoaded, которое поддерживается всеми современными браузерами и срабатывает после того, как будет построено дерево DOM. Функция будет выполнена до того, как будут загружены все внешние ресурсы, что делает страницу более отзывчивой. Internet Explorer до 9 версии не имел готового встроенного событияDOMContentLoaded, поэтому для того, чтобы заставить работать его так же, как и другие браузеры, приходилось искать обходные пути. В нашем примере на странице нет большого количества изображений, поэтому можно использовать подход с использованием события load – это не отразится на производительности страницы.

Теперь мы готовы к тому, чтобы привязать обработчик функции к событию щелчка по ссылке якоря. Когда пользователь щелкнет на якоре, будет выполнено определенное действие. В нашем примере будет создаваться новая заметка. Прежде всего необходимо пройти по дереву DOM до нашего конечного якоря, как показано в листинге 6.

**Листинг 6. Получение якоря с именем класса add**

load : function() {

var anchorSelected;

if (document.getElementsByClassName) {

anchorSelected = document.getElementsByClassName("add")[0];

} else {

var anchors = document.getElementsByTagName("a"),

alenght = anchors.length;

for (var i = 0; i < alenght; i++ ) {

var anchor = anchors[i];

if (anchor.className === "add") {

anchorSelected = anchor;

}

}

}

}

Как вы, вероятно, догадались, метод document.getElementsByClassName в листинге 6 позволяет получить элементы с указанным именем класса. Этот метод возвращает набор HTML-элементов, но, к сожалению, не полностью поддерживается всеми браузерами, например, IE6 и IE7. Для этих браузеров необходимо использовать другой код. Сначала можно получить список якорей с помощью метода document.getElementsByTagName, а затем пройти по этому списку в поисках якоря, содержащего класс CSS с именем add. Метод GetElementsByTagName формально возвращает объект NodeList и, к счастью, полностью поддерживается всеми основными браузерами.

Из листинга 6 видно, как сохранить размер массива якорей в переменной alength, чтобы впоследствии с помощью одного цикла for просмотреть все дерево DOM. Изменение дерева DOM и работа с ним – дорогостоящая операция, поэтому нужно стараться свести к минимуму число обращений к нему.

Теперь, получив якорь, мы можем привязать событие click к функции прослушивателя событий, ответственной за добавление заметки, как показано в листинге 7.

**Листинг 7. Привязка события**

load : function() {

...

SA.addEvent(anchorSelected, "click", SA.addNote, false);

}

Из листинга 7 видно, что прослушиватель событий, привязанный к событию click, называется SA.addNote. Эта функция предназначена для выполнения нескольких задач:

* Создание копии последней созданной заметки.
* Вставка текста, набранного пользователем, в только что созданную копию заметки.
* Добавление новой заметки в список заметок.

В листинге 8 показано решение первой задачи.

**Листинг 8. Создание копии последней созданной заметки**

addNote : function(event) {

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

// Clone the node

var newNode = notes.children[0].cloneNode(true);

},

Получив тег div с идентификатором заметки с помощью метода getElementById, мы получаем первый дочерний элемент, вложенный в div, и создаем его копию с помощью метода cloneNode. Полученную таким образом копию DOM-элемента мы сохраняем в переменную newNode.

Выберите элемент параграфа, вложенный в newNode, вызвав метод getElementsByTagName для клонированного элемента. Для получения содержимого узла в DOM предусмотрен атрибут textContent. К сожалению, он поддерживается полностью не всеми браузерами, поэтому придется использовать другой подход: обратитесь к атрибуту firstChild параграфа, а затем извлеките из него свойство nodeValue. Теперь значение этого свойства устанавливается равным содержимому тега textarea, который присутствует на странице. Содержимое тега textarea получается из значения свойства DOM-элемента textarea, доступного с помощью методаgetElementById. В листинге 9 показано, как вставить набранный пользователем текст в созданную копию заметки (вторая задача).

**Листинг 9. Вставка текста из текстового поля в только что созданную копию заметки**

addNote : function(event) {

...

// Set the content of the node

newNode.getElementsByTagName("p")[0].firstChild.nodeValue =

document.getElementById("contentArea").value;

notes.appendChild(newNode);

}

Для решения последней задачи мы добавим созданную заметку в список заметок с помощью метода appendChild, как показано в листинге 10.

**Листинг 10. Добавление новой заметки в список**

addNote : function(event) {

...

notes.appendChild(newNode);

}

Наконец, необходимо предотвратить выполнение действия по умолчанию для события click (для якоря этим действием является перенаправление пользователя по указанному в атрибуте href URL-адресу). Для этого в DOM предусмотрен методpreventDefault(), применяемый к событию с передаваемым в функцию-обработчик параметром. Этот метод также не поддерживается в IE версии ниже 9. Чтобы добиться тех же результатов при использовании более старых версий IE, можно установить атрибут event.returnValue в false. Код представлен в листинге 11.

**Листинг 11. Предотвращение выполнения действия по умолчанию для события click**

addNote : function(event) {

...

event.preventDefault ? event.preventDefault() : event.returnValue = false;

}

В листинге 12 приведен весь JavaScript-код файла script.js.

**Листинг 12. Содержимое файла script.js**

window.SA = {

addEvent : function(element, evType, fn, useCapture) {

if (element.addEventListener) {

element.addEventListener (evType, fn, useCapture);

return true;

} else if (element.attachEvent) {

var r = element.attachEvent('on' + evType, fn);

return r;

} else {

element['on' + evType] = fn;

}

},

load : function() {

var anchorSelected;

if (document.getElementsByClassName) {

anchorSelected = document.getElementsByClassName("add")[0];

} else {

var anchors = document.getElementsByTagName("a"),

alenght = anchors.length;

for (var i = 0; i < alenght; i++ ) {

var anchor = anchors[i];

if (anchor.className === "add") {

anchorSelected = anchor;

}

}

}

SA.addEvent(anchorSelected, "click", SA.addNote, false);

},

addNote : function(event) {

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

// Clone the node

var newNode = notes.children[0].cloneNode(true);

// Set the content of the node

newNode.getElementsByTagName("p")[0].firstChild.nodeValue

= document.getElementById("contentArea").value;

notes.appendChild(newNode);

event.preventDefault ? event.preventDefault() : event.returnValue = false;

}

}

SA.addEvent(window, "load", SA.load, false);

**Библиотеки JavaScript и модель DOM**

Когда разработчики пишут код на JavaScript, они часто используют библиотеки (инфраструктуры) JavaScript, которые самостоятельно учитывают различие реализаций модели DOM в различных браузерах. Посмотрите, как можно было бы переписать код JavaScript из листинга 12, используя популярную библиотеку jQuery.

**Листинг 13. Script-jquery.js**

$(function(){

$('a.add').click(function(){

var newNote = $('.note').eq(0).clone();

newNote.find('p').text($('#contentArea').val());

$('#notes').append(newNote);

return false;

});

});

В результате мы получили всего лишь несколько строк четкого и чистого кода.

Поскольку для использования jQuery необходимо импортировать библиотеку в HTML, как показано в листинге 14, придется выполнить один дополнительный HTTP–запрос. Кроме того, на выполнение библиотеки требуется больше времени. Этот процесс может замедлить работу приложения, поэтому вы должны самостоятельно найти разумный баланс между использованием библиотек и увеличением объема написанного кода.

**Листинг 14. Импорт библиотеки в HTML**

<html>

<head>

...

<script src="http://ajax.googleapis.com/ajax/libs/jquery

/1.6.1/jquery.min.js"></script>

</head>

Библиотеки JavaScript – очень мощный инструмент, способный значительно упростить вашу жизнь. Тем не менее необходимо иметь представление о DOM-скриптинге, поскольку использование библиотек не всегда является самым эффективным способом работы с DOM. Также советуем вам понять, что именно происходит в процессе работы библиотеки.

**Заключение**

Модель DOM очень важна для Web-разработчиков, поскольку она является механизмом взаимодействия JavaScript с Web-страницами. Существует целый ряд проблем и ограничений, связанных с реализацией API-интерфейсов DOM разработчиками различных браузеров. Некоторые атрибуты и методы поддерживаются полностью не всеми браузерами (например, addEventListener(),textContent), а в некоторых случаях ведут себя по-разному.

Еще один важный фактор, который необходимо учитывать при использовании DOM-скриптинга, – это производительность. Как было показано в этой статье, если вы знаете, как взаимодействуют JavaScript и DOM, то для управления и навигации по DOM можно использовать различные программные инфраструктуры JavaScript.