## Изменение дерева DOM

Как уже отмечалось выше, при анализе HTML кода браузер выстраивает древовидную структуру программных объектов, называемую «деревом DOM». Мы рассмотрели, как можно анализировать эту структуру, обращаться к соседним, родительским или дочерним элементам, получать их коллекции. В то же время дерево DOM допускает изменения — добавление или удаление его узлов. Рассмотрим эти процессы более детально.

Для внесения изменений в структуру дерева DOM предусмотрены следующие функции:

* removeChild(еl) — удаляет дочерний узел «еl», переданный как аргумент функции;
* appendChild(еl) — добавляет дочерний узел «еl» в конец существующей коллекции дочерних узлов;
* insertBefore(еl1, еl2) — вставляет узел «еl1» в коллекцию дочерних элементов перед узлом «еl2».

Эти функции присутствуют в каждом узле дерева DOM и позволяют оперировать с собственной коллекцией дочерних элементов. Если необходимо добавить или удалить элемент внутри дочернего элемента, следует перейти к нему и вызывать нужный метод у дочернего элемента. Аналогично, для управления соседями следует перейти к родительскому элементу и вызывать его методы управления дочерними элементами.

Для того чтобы добавить в дерево новый элемент, его необходимо сначала создать. Создаются новые узлы при помощи метода «createElement», принадлежащего объекту «document»:

document.createElement(tagName)

В качестве аргумента метод принимает имя тега для элемента, который будет создан. Имя указывается без угловых скобок «<>». Например, создать новый абзац (HTML тег <p>) можно командой

document.createElement("p")

Аналогично, для того чтобы создать блок (HTML тег <div>) следует применить команду document.createElement('div')

*Имена тегов при вызове метода заключаются в кавычки*. Вид кавычек, согласно стандартов JavaScript, роли не играет.

Продемонстрируем методы добавления узлов к дереву DOM на следующем примере. Создайте новый HTML файл, в котором наберите или скопируйте следующее содержимое: (*код также доступен в папке Sources — файл js4\_13.html*)

<html>

<head>

</head>

<body>

<ul id='list'>

<li>Item 1</li>

<li>Item 2</li>

<li>Item 3</li>

<li>Item 4</li>

</ul>

<button onclick="addItem()">Add item</button>

<button onclick="insertItem()">Insert item</button>

<script>

function addItem(){

var newItem = document.createElement('li');

newItem.innerText = "New item";

list.appendChild(newItem);

}

function insertItem(){

var firstItem = list.childNodes[0];

var newItem = document.createElement('li');

newItem.innerText = "New item";

list.insertBefore(newItem, firstItem);

}

</script>

</body>

</html>

Основу страницы составляет маркированный список <ul id='list'> с четырьмя пунктами, названными Item 1-4. После списка расположены две кнопки для демонстрации двух различных способов добавления новых элементов.

Нажатие первой кнопки вызывает функцию «addItem()». В теле этой функции происходит следующее:

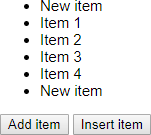
1. Создается новый элемент с именем тега «li» (именно такие элементы являются дочерними для списка) и помещается в переменную «newItem»: var newItem = document.createElement(‘li’);
2. Для нового элемента устанавливается текст при помощи инструкции:   
   newItem.innerText = "New item";
3. Новый элемент добавляется в коллекцию дочерних элементов списка. Напомним, что список доступен по имени своего идентификатора «id='list'»: list.appendChild(newItem)

Подобным образом работает и вторая функция «insertItem()», добавляющая новый элемент в начало списка при помощи метода «insertBefore». Поскольку для этого метода необходимо два аргумента, дополнительно определяется первый дочерний элемент списка путем обращения к коллекции дочерних элементов списка:

var firstItem = list.childNodes[0]

При вызове метода «insertBefore» указывается, что новый элемент необходимо вставить перед первым, то есть в начало списка.

Сохраните файл и откройте его в браузере. Нажимая на кнопки убедитесь, что новые элементы появляются как в начале, так и в конце списка.



*Рисунок 59*

При рассмотрении процесса удаления элементов из дерева DOM следует снова вспомнить об особенностях сборки этого дерева. Казалось бы, что нет ничего сложного и второй элемент из списка можно удалить одной инструкцией

list.removeChild( list.childNodes[1] )

Эта инструкция действительно удалит вторую строку списка, если

1. Второй элемент у списка вообще существует;
2. В списке нет дополнительных элементов, сбивающих нумерацию коллекции.

Условие 1 несложно проверить. Из первого урока мы знаем о том, что все неопределенные переменные имеют тип «undefined» и проверка наличия второго элемента в коллекции выглядит следующим образом:

if(typeof list.childNodes[1] !="undefined"){...}

Условие 2 возвращает нас к выводам, полученным в разделе 3 данного урока: в коллекции дочерних элементов списка, кроме элементов <li> могут присутствовать анонимные текстовые элементы, связанные с оформлением его HTML кода. Сложности добавляет тот факт, что эти элементы могут и не присутствовать, если оформление кода поменяется. То есть для того чтобы получить второй элемент списка (как мы его видим на странице) необходимо перебирать коллекцию его дочерних элементов и считать только те, которые отвечают за тег <li>. Остальные игнорировать.

Добавим к HTML коду нашей страницы еще одну кнопку, укажем функцию «removeItem» в качестве обработчика события ее нажатия

<button onclick="removeItem()"> Remove second item

</button>

В скриптовой части документа добавим эту функцию (код с изменениями доступен в папке Sources — файл js4\_14.html):

function removeItem(){

var n = 0;

var element2 = false;

for(var element of list.childNodes){

if(element.tagName == "LI") n++;

if(n==2) {

element2 = element;

break;

}

}

if(element2)

list.removeChild(element2);

}

В начале функции устанавливаем две переменные: счетчик строк — «n» и сам элемент, отвечающий за вторую строку «element2». Далее организовываем цикл «for-of», проходящий все элементы коллекции «list.childNodes».

В теле цикла проверяем отвечает ли данный элемент коллекции за тег «<li>». Для этого используем свойство «tagName», определяющее имя тега. Согласно спецификации JavaScript это имя хранится в верхнем регистре (большими буквами) и соответствует строке "LI". Если данный элемент коллекции имеет указанное имя, то увеличиваем счетчик строк «n++».

Затем, также в теле цикла, проверяем значение счетчика: если оно равно 2, то данный элемент цикла является нашим искомым — второй строкой списка. В таком случае сохраняем это значение в переменной «element2» и останавливаем цикл. В противном случае цикл перейдет к следующей итерации.

После окончания цикла проверяем, был ли вообще найден второй элемент. Если был, то вызываем метод «list.removeChild» для него.

Сохраните файл и обновите страницу браузера. Убедитесь в работоспособности новой кнопки, а также в отсутствии ошибок, когда в списке остается только одна строка.

**Задание для самостоятельной работы:** дополните созданную программу кнопкой, по нажатию на которой новый элемент будет добавляться в центр списка (или перед центральным элементом, если их количество нечетное).

Рассмотрим еще один пример, иллюстрирующий манипуляции с деревом DOM. Поставим себе задачу: разработать список, в котором можно менять порядок элементов при помощи технологии Drag-and-Drop, то есть перетягивать элементы списка мышкой и вставлять их в нужное место списка.

Технологию Drag-and-Drop мы рассматривали выше. Для нашей задачи ее надо немного видоизменить. Когда пользователь нажимает на некоторый пункт списка, он не должен исчезнуть из списка при перетягивании. Иначе возникнет сбой нумерации списка и скачок его размера. Также может быть не совсем понятно, в какое место спи- ска элемент вставится, если его отпустить.

Сделаем так:

* При нажатии кнопки мыши и начале перетягивания элемент в списке получит некоторое выделение для того, чтобы понятно было какой элемент является активным. Из списка он не исчезнет, чем сохранит нумерацию и размеры всего списка.
* Вместе с курсором мыши будет двигаться копия активного элемента, создавая видимость процесса перетягивания.
* При смене позиции курсора мыши выделенный (активный) элемент списка будет менять свое место в списке, следуя за курсором.

Приведем полный код документа и далее проведем анализ его работы. Создайте новый HTML файл, в котором наберите или скопируйте следующее содержимое (*код также доступен в папке Sources — файл js4\_15.html*):

<!doctype html>

<html>

<head>

<style>

body{

position: relative;

}

.item, .phantom { height: 30px;

width: 300px;

}

.phantom{

position: absolute;

}

</style>

</head>

<body>

<ol id='list'>

<li class='item' style='background-color:gold'></li>

<li class='item' style='background-color:red'></li>

<li class='item' style='background-color:green'></li>

<li class='item' style='background-color:blue'></li>

</ol>

<script>

var draggedElement = false;

var phantomElement = false;

document.onmousedown = function(e){ e.preventDefault();

var clickedElement = document. elementFromPoint(e.clientX, e.clientY); if(clickedElement.className.indexOf('item')>-1){

clickedElement.style.opacity = '0.5';

draggedElement = clickedElement;

}

}

document.onmousemove = function(e){ if(draggedElement){

if(!phantomElement){ phantomElement =

document.createElement('div'); phantomElement.style.backgroundColor =

draggedElement.style. backgroundColor;

phantomElement.style.left =

e.pageX-draggedElement. offsetWidth/2 + 'px';

phantomElement.style.top =

e.pageY-draggedElement. offsetHeight/2 + 'px';

phantomElement.className =

"phantom"; document.body.appendChild(

phantomElement);

}

else

phantomElement.style.left =

e.pageX-phantomElement. offsetWidth/2 + 'px';

phantomElement.style.top =

e.pageY-phantomElement. offsetHeight/2 + 'px';

phantomElement.style.zIndex = '-1'; var lowerElement = document.

elementFromPoint(e.clientX,

e.clientY); phantomElement.style.zIndex = '1';

if(lowerElement != null

&& lowerElement != draggedElement && lowerElement.className. indexOf('item')>-1){

if(lowerElement == window.list.lastChild &&

e.pageY > (lowerElement.offsetTop+ lowerElement.offsetHeight/2)){

window.list. removeChild(draggedElement);

window.list. appendChild(draggedElement);

}

else

{

if(e.pageY > (lowerElement.offsetTop + lowerElement.offsetHeight/2)){

if(lowerElement. previousSibling == draggedElement){

window.list. removeChild( draggedElement);

window.list.insertBefore( draggedElement, lowerElement. nextSibling);

}

}

else

{

window.list.removeChild(

draggedElement); window.list.insertBefore(

draggedElement, lowerElement);

}

}

}

}

}

}

document.onmouseup = function(e){ if(draggedElement){

draggedElement.style.opacity = '1'; draggedElement = false;

}

if(phantomElement){ document.body.removeChild(phantomElement); phantomElement = false;

}

}

</script>

</body>

</html>

Основу HTML части составляет список «<ol id='list'>» в котором создано четыре элемента. Эти элементы выделены разными цветами для большей наглядности. Список является нумерованным, что позволит нам следить за тем, что количество и нумерация элементов при перетягивании не меняется. Внешний вид списка на вкладке браузера представлен на следующем рисунке:



*Рисунок 60*

Далее следует скриптовая часть. Для того чтобы обеспечить функциональность технологии Drag-and-Drop мы вводим глобальные переменные:

var draggedElement = false; var phantomElement = false;

Первая «draggedElement» будет отвечать за реальный элемент списка, который будет перемещаться — за активный элемент. Вторая «phantomElement» будет ссылаться на дополнительный элемент — копию активного элемента списка, который будет следовать за указателем мыши.

С целью отделения кода JavaScript и разметки HTML обработчик события нажатия кнопки мыши реализован у объекта «document». Ранее мы приводили аргументы в пользу того, что данный обработчик логично реализовывать у перетягиваемого объекта. Однако в нашей новой задаче перетягиваемых объектов несколько. Более того, их количество может меняться, если в список будут добавляться или удаляться элементы. Вместо того чтобы перебирать все эти элементы и устанавливать для них отдельные обработчики создадим универсальный метод, подходящий для всех элементов.

В таком случае он должен принадлежать одному из родительских элементов, общим для который является «document»:

document.onmousedown = function(e)

Следует отметить, что браузеры сами по себе могут поддерживать технологию Drag-and-Drop, позволяя перетянуть элементы из браузера в проводник или рабочий стол, обеспечивая таким образом быстрое сохранение картинок или других объектов веб-страницы. Для того чтобы наша задача не конфликтовала со встроенными механизмами браузера, первым делом отменим стандартную обработку события нажатия кнопки мыши командой «e.preventDefault()».

Далее нам необходимо определить элемент списка, находящийся в точке под курсором мыши. Поскольку событие получает не сам элемент, а документ, это необходимо сделать в два этапа:

* Определяем элемент под курсором мыши и сохраняем его в переменной «clickedElement». Используем стандартный метод «elementFromPoint» объекта «document»: var clickedElement = document.elementFromPoint(e.clientX, e.clientY);
* Проверяем, является ли данный объект элементом списка, т.к. документ будет получать события и от совершенно других элементов. Делаем это путем проверки наличия класса «item» у определенного объекта: if(clickedElement.className.indexOf(‘item’) > -1);
* Обратите внимание, что все элементы списка подключают этот стилевой класс при HTML объявлении.

В случае, если проверка проходит успешно, выделяем данный элемент, изменяя в два раза его прозрачность:

clickedElement.style.opacity = ‘0.5’;

А также сохраняем данный элемент в глобальной переменной для обеспечения возможности доступу к нему из других функций, в частности, из обработчика событий движения мыши:

draggedElement = clickedElement;

Далее рассмотрим основную функцию — обработчик движения мыши. Поскольку все действия должны выполняться только в том случае, если происходит процесс перетягивания, все тело функции заключено в соответствующее условие:

document.onmousemove = function(e){ if(draggedElement){

Затем проверяем наличие элемента-копии (фантома), который следует за курсором мыши. При нажатии кнопки мыши мы его не создавали, т.к. щелчок мышью еще не означает перетягивание. Анализируем это именно при движении мыши.

Если фантома нет, то создаем его как отдельный блок «div»:

phantomElement = document.createElement('div');

Мы не должны добавлять фантом как элемент списка, иначе количество элементов списка увеличится. Фантом не будет принадлежать списку, играя роль лишь визуализации перетягивания.

После создания нового элемента устанавливаем для него такой же цвет, как у активного элемента в списке, чтобы дополнительно информировать пользователя какой элемент перемещается:

phantomElement.style.backgroundColor = draggedElement.style.backgroundColor

Далее устанавливаем координаты блока учитывая координаты курсора мыши. По аналогии с предыдущими примерами по технологии «Drag-and-Drop» вы наверняка заметили, что блок смещается таким образом, чтобы курсор мыши находился в его центре (смещается на поло- вину ширины и высоты). Остальные стилевые атрибуты для блока-фантома задаются при помощи подключения стилевого класса «phantom». После чего блок добавляется к дочерним элементам тела документа:

document.body.appendChild(phantomElement)

Еще раз повторимся, блок не должен принадлежать списку, чтобы не влиять на его структуру. Фантом принадлежит телу документа.

Вторая часть условного оператора (else) отвечает за ситуацию, когда фантомный элемент уже был создан ранее в предыдущих вызовах события. Первым делом для фантома устанавливаются новые координаты, согласно координатам курсора мыши, переданным в аргументе события. Затем определяется элемент списка, находящийся под курсором мыши. Здесь тоже есть своя особенность: для того чтобы фантомный элемент был «поверх» остальных элементов списка ему установлен стилевой атрибут «z-index». Однако, в таком случае именно он всегда будет тем элементом, который находится под курсором мыши. Поэтому элемент списка мы определяем по следующему алгоритму:

1. Прячем фантомный элемент ниже списка, устанавливая отрицательное значение атрибута «z-index»:

phantomElement.style.zIndex = '-1'

1. Определяем элемент, находящийся под курсором мыши уже известной нам командой «elementFromPoint». Результат сохраняем в переменной «lowerElement»:

var lowerElement = document.elementFromPoint(e.clientX, e.clientY);

1. Возвращаем фантомному элементу исходное значение атрибута «z-index» чтобы он и далее отображался поверх остальных элементов:

phantomElement.style.zIndex = '1';

В результате мы получим элемент, находящийся под курсором мыши и под фантомным элементом, т.к. в момент определения мы перемещали его в нижний слой разметки страницы.

Далее необходимо проверить, что элемент под курсором вообще существует. Если движение мыши происходит по пустой части страницы, то такого элемента не будет и в переменной «lowerElement» сохранится значение «null». Также следует убедится, что данный элемент принадлежит списку. Это мы уже разбирали — достаточно проверить наличие у элемента стилевого класса «item». Дополнительно нужно проверить то, что курсор мыши перешел к следующему элементу списка, а не находится еще в пределах активного элемента.

В итоге, комплексная проверка на то что элемент под курсором «lowerElement» существует, этот элемент относится к списку и не является активным для перетягивания будет реализована условием:

if(lowerElement != null && lowerElement != draggedElement && lowerElement.className.indexOf('item')>-1){

Если все эти условия выполнены, то курсор мыши перешел к другому (неактивному) элементу списка, а значит необходимо поменять местами в списке элементы, хранимые в переменных «lowerElement» и «draggedElement». Если реализовать перестановку элементов непосредственно, то возможно появление эффекта «дребезга» — попеременная перестановка элементов списка в случаях, когда курсор мыши находится на границе между соседними элементами. Незначительные движения мыши на такой границе будут приводить к тому, что курсор находится то над одним элементом, то над другим. Возникнет неприятное мерцание со сменой элементов друг друга.

Для того чтобы нивелировать эффект дребезга добавим дополнительное условие — курсор мыши должен не просто перейти к соседнему элементу (превысив значение «lowerElement.offsetTop»), а пройти дальше, чем находится его центр по высоте («lowerElement.offsetHeight/2»). Это дополнительное условие будет выражено как

e.pageY > (lowerElement.offsetTop + lowerElement.offsetHeight/2)

Координата Х курсора мыши на перестановку списка не влияет, т.к. список расположен вертикально. Соответственно, ее значение в условиях не учитывается.

Перестановка местами элементов «lowerElement» и «draggedElement» также будет отличаться для различных их взаимных расположений. Если «lowerElement» является последним в списке, то для перестановки нужно элемент «draggedElement» добавить в конец списка. Это обеспечивается методом «appendChild».

Если мышь движется вверх и активный элемент перемещается ближе к началу списка, то «draggedElement» должен быть вставлен в список непосредственно перед «lowerElement». Такую перестановку обеспечит вызов метода «insertBefore(draggedElement,lowerElement)».

Если движение происходит в обратном направлении, то «draggedElement» необходимо вставлять в список после «lowerElement». Однако такого метода как «вставить после» в модели DOM не предусмотрено. Воспользовавшись знаниями об отношениях узлов дерева DOM отметим, что «вставить после узла» это значит «вставить перед следующим соседом узла». То есть вызов метода примет вид:

insertBefore(draggedElement,lowerElement.nextSibling)

Если же у элемента нет следующего соседа, то значит элемент является последним, а эту ситуацию мы разобрали выше отдельным пунктом обсуждения.

Отметим, что перед вставкой элемента «draggedElement» в список в новую позицию, его необходимо удалить в старой позиции командой «removeChild(draggedElement)». Это касается всех трех вариантов перестановки. Также отметим, что все команды управления деревом DOM должны вызываться у списка, т.к. речь идет о перестановках его дочерних элементов. То есть все описанные выше команды относятся к объекту «window.list».

В завершение, рассмотрим обработчик события от- пускания кнопки мыши. Он состоит из двух условных операторов, определяющих были ли ранее созданы объекты «draggedElement» и «phantomElement». Напомним, что они создаются раздельно и, следовательно, требуют отдельных проверок.

Элемент «draggedElement» отвечает за реальный объект в списке, позиция которого меняется при движении мыши. Окончание перетягивания, наступающее после отпускания кнопки мыши, должно снять с этого элемента выделение. При нажатии кнопки мыши мы уменьшали прозрачность элемента. Значит вернем ее в исходное значение.

draggedElement.style.opacity = '1';

Затем установи значение «false» для переменной «draggedElement» для того чтобы обработчик события движения мыши не выполнял команды по перемещению элементов списка. Удалять объект «draggedElement» не нужно, т.к. при этом изменится сам список.

Элемент «phantomElement», наоборот, создан в документе дополнительно, а значит по завершению перетягивания должен быть удален из тела документа:

document.body.removeChild(phantomElement);

Полностью аналогично, самой переменной «phantom- Element» также устанавливается значение «false».

Разобравшись в алгоритме работы программной части, сохраните файл и откройте его в браузере. Внешний вид списка должен соответствовать приведенному ранее рисунку.

Наведите курсор мыши на любой элемент списка и нажмите кнопку мыши. Цвет элемента должен поблекнуть. Не отпуская кнопку начните движение мыши. Возле ее курсора появится блок с теми же размерами и цветом, как у активного элемента списка. Проведите мышью вверх и вниз списка, обратите внимание как происходит смена позиции активного элемента.



*Рисунок 61*

Отпустите кнопку мыши, убедитесь в том, что фантомный элемент исчезает, активный элемент восстанавливает насыщенность цвета и новый порядок списка остается неизменным. Повторите описанные действия с другими элементами списка, устанавливая их в произвольном порядке.

**Задание для самостоятельной работы:** реализуйте в программе с перестановкой списка улучшение, позволяющее «тянуть» выбранный элемент за ту же точку, на которой произошел щелчок мыши, а не только за центр блока (воспользуйтесь тем же методом, что и при решении задачи из примера «Drag-and-Drop»).