Урок 5

Введение в DOM

Модель документа и работа с ней при помощи JavaScript.

Введение

На предыдущих уроках мы работали с консолью и всплывающими подсказками. Хоть мы и продвинулись в изучении языка JavaScript, нам не хватает большого пласта знаний, касающихся взаимодействия наших скриптов со страницей, на которой они запущены. Без них мы не сможем писать интерактивные динамические страницы, которые могут менять свой вид уже после полной загрузки на стороне клиента.

DOM

Понятие DOM

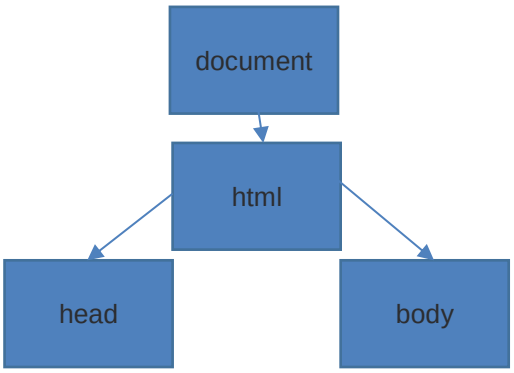
При работе на стороне клиента мы можем разделить рабочие объекты на две сущности: JavaScript и HTML. Первая отвечает за некую динамическую логику, а вторая – за разметку. Их взаимодействие происходит через специальный объект, называемый объектной моделью документа (DOM – Document Object Model). DOM генерируется в момент загрузки HTML-документа браузером.

1. В момент загрузки браузер читает разметку HTML.
2. Во время чтения создаётся набор сущностей, которые связаны между собой. Связи эти образованы на основании верстки.
3. Сущности сохраняются в модели DOM.
4. Наш код на JS общается с DOM, обращаясь к элементам вёрстки и их содержимому.
5. Когда наш код меняет структуру DOM, браузер автоматически обновляет (но не перезагружает!) страницу, показывая новую структуру и содержимое.

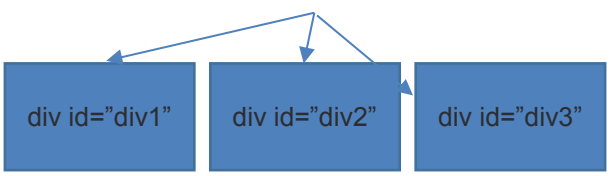
С точки зрения JS структура DOM хранится в глобальном объекте document. Внутри него хранится иерархия согласно вложенности тегов в вёрстке, т.е.

- document
  - html
    - head
      - title
      - meta
    - body

Теперь попробуем попрактиковаться в добывании тех или иных элементов из DOM. Возьмём простую структуру



Ей будет соответствовать вот такой HTML-код:



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>My Page</title>
</head>
<body>
  <h2 id="div1">Some data in div-1</h2>
  <h2 id="div2">Another data in div-2</h2>
  <h2 id="div3">Anything else in div-3</h2>
</body>
</html>
```

Редактирование DOM

При помощи JavaScript мы можем менять и дополнять нашу страницу. Для начала возьмём и заменим содержимое “Another data in div-2” на “Совершенно новая информация”. Для этого нам понадобится элемент с идентификатором “div-2”.

```
var content2 = document.getElementById("div-2");
```

Метод getElementById у объекта document возвращает нам элемент страницы, который соответствует указанному идентификатору (атрибуту id). После этого JavaScript получает возможность работать с этим элементом и менять его.
Для изменения содержимого мы используем свойство innerHTML у нашего объекта content2. Оно позволяет нам изменять содержимое указанного элемента.

```
content2.innerHTML = "Совершенно новая информация";
```

Готово. При загрузке страницы содержимое div-2 заменится на нужный нам текст. Давайте подробно разберём, что же мы на самом деле сделали.

При помощи объекта document мы получили доступ к DOM внутри JavaScript. Метод объекта document по полученному идентификатору вернул нам нужный элемент. Примерно таким же образом происходит обращение к элементам в CSS.

Методы объекта element

Результатом работы метода getElementById является объект element, применяемый для чтения и изменения содержимого элемента. И главное для нас преимущество тут в том, что при этом изменяется и страница, на которой запущен скрипт, причём делает она это без перезагрузки.

Свойство объекта element под названием innerHTML, как Вы уже увидели может применяться как для чтения, так и для редактирования содержимого выбранного элемента. innerHTML возвращает внутреннее содержимое элемента в виде строкового значения, исключая теги HTML. При попытке получить элемент из DOM по идентификатору, которого на странице (т.е. в DOM) нет, метод

getElementById вернет null. Поэтому при использовании getElementById нужно выполнять проверку на null до того, как начнётся работа со свойствами ожидаемого элемента.
Наряду с getElementById есть также и метод getElementsByClassName, позволяющий получать элементы по имени класса. Работает он примерно также, но на выходе Вы получите набор элементов (коллекцию - структуру данных, схожую с массивом, но отличающуюся от него), подходящих под условия выборки. Ведь на странице может содержаться множество элементов с одинаковым классом. Также существует метод, который возвращает набор элементов — getElementsByTagName, который возвращает все элементы с указанным именем тега.

Момент загрузки кода

Работая с DOM очень важно помнить весьма существенный нюанс. Код скрипта должен выполняться только после полной загрузки страницы. В случае, если код стартует, не дожидаясь загрузки, скорее всего модель DOM не будет полностью построена. Таким образом, Вы будете пытаться работать с ожидаемыми, но пока несуществующими элементами.

Вы, разумеется, можете сказать: «А почему бы не поместить код в конец HTML-документа? Ведь страница генерируется сверху вниз, и у нас будет гарантия того, что DOM к этому времени уже будет сформирована». Совершенно верное рассуждение, но есть более изящный и надежный способ гарантировать старт скриптов только после полной загрузки страницы.

Необходимо создать функцию, которая содержит наш код. Она должна выполняться после загрузки страницы. Это обеспечивается присвоением данной функции свойству onload у объекта window. Сам по себе объект window является встроенным глобальным объектом в JS. Объект устроен таким образом, что вызовет любую функцию, указанную в свойстве onload, но только после того, как страница будет полностью загружена.

```
function my_initiation() {
  var content2 = document.getElementById("div-2");
  content2.innerHTML = "Совершенно новая информация";
}
Window.onload = my_initiation;
```

Здесь стоит обратить внимание на то, что после имени функции круглые скобки не ставятся. Ведь мы здесь не пытаемся вызывать эту функцию, а просто указываем её имя для того, чтобы она могла быть вызвана при загрузке страницы.

Создание элемента

Теперь рассмотрим ситуацию, при которой нам необходимо создать совершенно новый элемент в структуре DOM прямо в JS коде. Для этого нам потребуется метод объекта document – createElement. Он создает новый элемент с указанным тегом.

```
var div = document.createElement("div");
```

Помимо этого, мы можем дать ему классы и заполнить текстом. Все это – свойства объекта element. Таким образом, мы можем их задавать простым присвоением.

```
var div = document.createElement("div");
```

```
div.className = "my_div";
div.innerHTML = "<h1>Заголовок</h1><p>Содержимое</p>;
```

Но пока указатель на наш новый элемент всего лишь сохранён в переменной div, на странице он не виден. Для того, чтобы новый элемент появился на странице, его необходимо туда поместить, говоря иначе – добавить его к объекту document.
Для начала решим, в какое место документа будет размещён наш элемент. Если у Вас есть некий родительский элемент, готовый принять новый div (а он у Вас есть – как минимум, это body), то для вставки можно применять следующие методы.
1. parentElem.appendChild(elem) – он добавляет элемент elem в конец дочерних элементов parentElem.

```
<ol id="list">
<li>0</li>
<li>1</li>
<li>2</li>
</ol>
<script>
var newLi = document.createElement("li");
newLi.innerHTML = "Привет, мир!";
list.appendChild(newLi);
</script>
```

2. parentElem.insertBefore(elem, nextSibling)

Вставляет elem в коллекцию детей parentElem, перед элементом nextSibling.

```
<ol id="list">
<li>0</li>
<li>1</li>
<li>2</li>
</ol>
<script>
var newLi = document.createElement("li");
newLi.innerHTML = "Привет, мир!";
list.insertBefore(newLi, list.children[1]);
</script>
```

3. parentElem.removeChild(elem)
Удаляет elem из списка детей parentElem.

4. parentElem.replaceChild(newElem, elem)
Среди детей parentElem удаляет elem и вставляет на его место newElem.
Методы 3 и 4 возвращают удалённый узел, что позволяет вставить его в другое место документа при необходимости.

Управление атрибутами

Мы научились управлять содержимым тегов, теперь попробуем задать атрибут элемента из кода нашего скрипта. Это может пригодиться, например, если Вы хотите присваивать классы элементам динамически. Это полезно в случае, когда надо выделять ту или иную сгенерированную на стороне клиента информацию. Допустим, мы хотим подсветить наш новый текст синим цветом. Тогда можно взять и назначить нужный нам класс .blue_color { color: blue; } для выбранного элемента.
Объект element содержит метод под названием setAttribute. Он может вызываться для создания или изменения атрибутов элементов HTML-кода.

```
content2.setAttribute("title", "My title");
```

Метод получает два аргумента: имя редактируемого атрибута и его значение. Если до вызова метода атрибута у элемента не существовало, то он создастся в момент выполнения.
Если мы хотим изменить стили элемента, то можно использовать атрибут style. Он предоставляет доступ к набору CSS-свойств выбранного элемента. Например:

```
content2.style.color = "blue";
```

Однако, несмотря на возможность менять стили, делать этого не рекомендуется, т.к. это меняет inline-стили. А это, как Вы уже знаете из курса HTML/CSS, не самая хорошая практика, и рекомендуется пользоваться классами для задания стилей.
Наряду с изменением атрибутов есть и метод чтения атрибутов. Это метод getAttribute, который вызывается для получения значения атрибута элемента HTML-кода.

```
var class_value = content2.getAttribute("class");
```

Если искомый атрибут не задан, то метод вернёт null.
Также есть свойство className, которое отвечает за значение атрибута class выбранного элемента. Оно позволяет задавать строку классов (через пробел) или получать классы. У него есть более мощный ”старший брат” - classList, который возвращает все классы выбранного элемента в виде структуры DomTokenList.

```
<div id="clock" class="example for you"> </div>
```

```
var elem = document.querySelector("#clock")
// Выведем классы
console.log(elem.classList); // DOMTokenList ["example", "for", "you"]
// Добавим классы
elem.classList.add("ok", "understand");
console.log(elem.classList);                    // DOMTokenList ["example", "for", "you", "ok",
"understand"]
// Переключим классы
elem.classList.toggle("you");
elem.classList.toggle("he");
console.log(elem.classList);                    // DOMTokenList ["example", "for", "ok", "understand",
"he"]
// Проверим класс
console.log(elem.classList.contains("example")); // true
console.log(elem.classList.contains("lol"));          // false
// И удалим классы
elem.classList.remove("example", "for", "understand", "he");
console.log(elem.classList);                     // DOMTokenList ["ok"]
```

Другие возможности по работе с DOM

Разумеется, возможности по управлению моделью DOM совсем не ограничиваются перечисленными выше методами. Многие из них мы встретим далее в процессе изучения JS. В общем и целом, JavaScript предоставляет следующий перечень методов для работы с DOM.

- Получение элементов из DOM – язык позволяет выбирать элементы страницы не только по id. К примеру, есть возможность управлять данными, которые пользователь вводит в формы обратной связи, или же обращаться к элементам через атрибуты, отличные от id.
- Создание и добавление элементов в DOM – и любые изменения сразу же отразятся на странице.
- Удаление элементов из DOM – то же самое, только в обратную сторону.
- Перебор элементов в DOM – после получения доступа к выбранному элементу, Вы можете найти все его дочерние элементы, «соседей», и т.д.

Практикум. Квест 2.0

Вспомним нашу игру из 4 урока. В ней чего-то не хватает, не так ли? А именно, истории вопросов и ответов. Теперь, когда мы умеем управлять DOM, мы можем выводить историю прямо на экран. Давайте сделаем это улучшение.

Домашнее задание

1. Создать функцию, генерирующую шахматную доску. При этом можно использовать любые html-теги по своему желанию. Доска должна быть разлинована соответствующим образом, т.е.
   чередовать черные и белые ячейки. Строки должны нумероваться числами от 1 до 8, столбцы – латинскими буквами A, B, C, D, E, F, G, H.
2. Сделайте генерацию корзины динамической, т.е. вёрстка корзины не должна находиться в HTML структуре. Там должен быть только div, в который будет вставляться корзина, сгенерированная на базе JS
   a. Пустая корзина должна выводить строку “Корзина пуста”
   b. Наполненная корзина должна выводить “В корзине: n товаров на сумму m рублей”
3. * Сделайте так, чтобы товары в каталоге выводились при помощи JS, т.е.
   a. В начале создайте массив товаров (сущность “Product”)
   b. При загрузке страницы на базе данного массива генерируйте вывод из массива. HTML-код должен содержать только div id=”catalog” без вложенного кода. Весь вид каталога генерируется JS

Дополнительные материалы

1. http://frontender.info/dom/ - что такое DOM
2. https://habrahabr.ru/post/312022/ - JavaScript и тренды 2016

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. «JavaScript. Подробное руководство» - Дэвид Флэнаган
2. «Изучаем программирование на JavaScript» - Эрик Фримен, Элизабет Робсон