Урок 5

Введение в DOM

Модель документа и работа с ней при помощи JavaScript.

DOM

- Понятие DOM
- Редактирование DOM
- Методы объекта element
- Момент загрузки кода
- Создание элемента
- Управление атрибутами
- Другие возможности по работе с DOM

Практикум. Квест 2.0

Практическое задание

Дополнительные материалы

Используемая литература

Введение

На предыдущих уроках мы работали с консолью и всплывающими подсказками. Теперь изучим взаимодействие скриптов со страницей, на которой они запущены. Это нужно, чтобы писать интерактивные динамические страницы, которые могут менять свой вид после полной загрузки на стороне клиента.

DOM

Понятие DOM

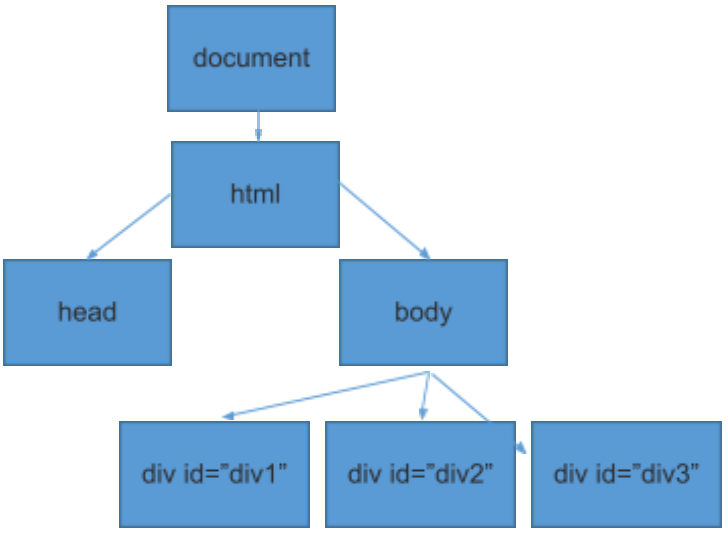
При работе на стороне клиента можно разделить рабочие объекты на две сущности: JavaScript и HTML. Первая отвечает за динамическую логику, вторая — за разметку. Их взаимодействие происходит через специальный объект — объектную модель документа (**DOM** — Document Object Model). DOM генерируется в момент загрузки HTML-документа браузером.

1. В момент загрузки браузер читает разметку HTML.
2. Во время чтения создается набор сущностей, которые связаны между собой на основании верстки.
3. Сущности сохраняются в модели DOM.
4. Код на JS общается с DOM, обращаясь к элементам верстки и их содержимому.
5. Когда 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 находится иерархия согласно вложенности тегов в верстке:

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

Попрактикуемся: будем добывать элементы из DOM. Возьмем простую структуру:



Ей будет соответствовать 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 можно менять и дополнять страницу. Заменим содержимое ***Another data in div-2*** на «Совершенно новая информация». Для этого понадобится элемент с идентификатором **div-2**.

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

Метод **getElementById** у объекта **document** возвращает элемент страницы, который соответствует указанному идентификатору (атрибуту **id**). После этого JavaScript может работать с этим элементом и менять его.

Чтобы изменить содержимое указанного элемента, используем свойство **innerHTML** у объекта **content2**.

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

При загрузке страницы содержимое **div-2** заменится на нужный текст. Разберем подробнее: при помощи объекта **document** мы получили доступ к **DOM** внутри JavaScript. Метод объекта **document** по полученному идентификатору вернул нужный элемент. Примерно так же происходит обращение к элементам в CSS.

Методы объекта element

Результат работы метода **getElementById** — объект **element**, который применяется для чтения и изменения содержимого элемента. Преимущество в том, что при этом без перезагрузки изменяется и страница, на которой запущен скрипт.

Свойство объекта **element** — **innerHTML** — 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 нет на странице (то есть в **DOM**), метод **getElementById** вернет **null**. Поэтому при использовании **getElementById** нужно выполнять проверку на **null** до того, как начнется работа со свойствами ожидаемого элемента.

Наряду с **getElementById** есть метод **getElementsByClassName**, позволяющий получать элементы по имени класса. Работает он примерно так же, но на выходе дает коллекцию элементов (структуру данных, схожую с массивом), подходящих под условия выборки. Ведь на странице может содержаться множество элементов с одинаковым классом. Еще есть метод **getElementsByTagName**, который возвращает набор элементов с указанным именем тега.

Момент загрузки кода

Работая с **DOM**, важно помнить, что код скрипта должен выполняться только после полной загрузки страницы. Если он стартует, не дожидаясь ее, — скорее всего, модель **DOM** не будет полностью построена и вы будете пытаться работать с ожидаемыми, но пока не существующими элементами.

Вы можете сказать: «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». Рассуждение верное, 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. Для этого надо присвоить эту функцию свойству **onload** у объекта **window** — встроенного глобального объекта в JS. Он 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;
```

Обратите внимание: после имени функции круглые скобки не ставятся. Мы не пытаемся вызывать эту функцию, а просто указываем ее имя, чтобы ее можно было вызвать при загрузке страницы.

Создание элемента

Научимся создавать новый элемент в структуре **DOM** прямо в JS-коде. Для этого потребуется метод объекта **document** — **createElement**. Он создает новый элемент с указанным тегом.

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

Можно дать ему классы и заполнить текстом. Все это — свойства объекта **element**. Их можно задавать простым присвоением.

```
var div = document.createElement("div");
div.className = "my_div";
div.innerHTML = "<h1>Заголовок</h1><p>Содержимое</p>;
```

Но пока указатель на новый элемент всего лишь сохранен в переменной **div**. Чтобы элемент появился на странице, его необходимо туда поместить — добавить его к объекту **document**.

Решим, в какое место документа разместить элемент. Если есть родительский элемент, готовый принять новый **div** (а он есть — как минимум это **body**), то для вставки можно применять следующие методы:

1. **parentElem.appendChild(elem)** — 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**.

```
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 удаленный узел, что позволяет вставить его в другое место документа при

необходимости.

Управление атрибутами

Зададим атрибут элемента из кода скрипта. Это может пригодиться, чтобы присваивать классы элементам динамически или выделять сгенерированную на стороне клиента информацию. Чтобы подсветить новый текст синим цветом, можно назначить нужный класс **.blue_color { color: blue; }** для выбранного элемента.

Объект **element** содержит метод **setAttribute**. Его вызывают, чтобы создавать или изменять атрибуты элементов HTML-кода.

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

Метод получает два аргумента: имя редактируемого атрибута и его значение. Если до вызова метода у элемента не существовало атрибута, то он создастся в момент выполнения.

Чтобы изменить стили элемента, можно использовать атрибут **style**. Он предоставляет доступ к набору CSS-свойств выбранного элемента:

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

Есть возможность менять стили, но делать это не рекомендуется, так как изменения претерпевают inline-стили. Лучше задавать стили с помощью классов.

getAttribute — метод чтения атрибутов, который вызывается для получения значения атрибута элемента HTML-кода.

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

Если искомый атрибут не задан, метод вернет **null**.

Свойство **className** отвечает за значение атрибута **class** выбранного элемента. Оно позволяет задавать строку классов (через пробел) или получать их. Более мощный аналог — **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

Возможности по управлению моделью DOM не ограничиваются перечисленными методами. Многие из них разберем при дальнейшем изучении JS. Обобщим методы для работы с DOM, которые предоставляет JavaScript:

1. **Получение элементов из DOM** — язык позволяет выбирать элементы страницы не только по **id**. Можно управлять данными, которые пользователь вводит в формы обратной связи, или обращаться к элементам через атрибуты, отличные от **id**.
2. **Создание и добавление элементов в DOM** — любые изменения сразу отразятся на странице.
3. **Удаление элементов из DOM** — то же самое, только в обратную сторону.
4. **Перебор элементов в DOM** — получив доступ к выбранному элементу, можно найти все его дочерние элементы или «соседей».

Практикум. Квест 2.0

Вернемся к игре из четвертого урока — ей нужна история вопросов и ответов. Теперь, когда умеем управлять DOM, можем выводить историю на экран. Реализуем это улучшение.

Практическое задание

1. Создать функцию, генерирующую шахматную доску. Можно использовать любые html-теги. Доска должна быть верно разлинована на черные и белые ячейки. Строки должны нумероваться числами от 1 до 8, столбцы — латинскими буквами A, B, C, D, E, F, G, H.
2. Сделать генерацию корзины динамической: верстка 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 должна выводить «В корзине: **n** товаров на сумму **m** рублей».
3. * Сделать так, чтобы товары в каталоге выводились при помощи JS:
    a. Создать массив товаров (сущность **Product**);
    b. При загрузке страницы на базе данного массива генерировать вывод из него. HTML-код должен содержать только **div id=”catalog”** без вложенного кода. Весь вид каталога генерируется JS.

Дополнительные материалы

1. Что такое DOM.
2. JavaScript и тренды 2016.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

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