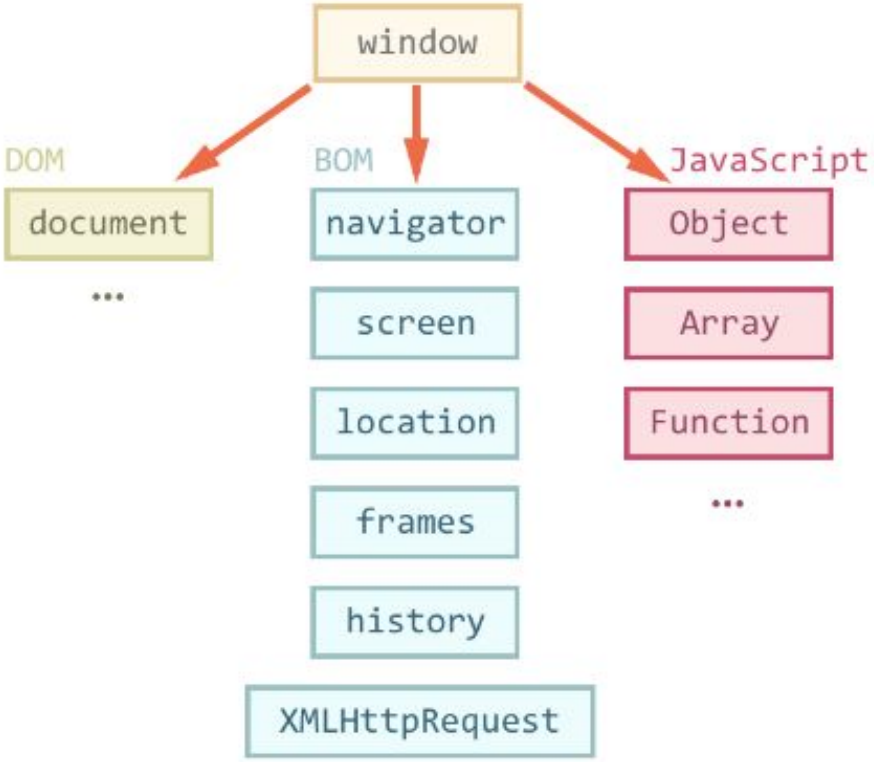
Окружение: DOM, BOM и JS



Объектная модель документа (DOM)

Глобальный объект **document** даёт возможность взаимодействовать с содержимым страницы.

```
document.body.style.background = 'red';
alert( 'Элемент BODY стал красным, а сейчас обратно вернётся' );
document.body.style.background = '';
```

Дерево DOM

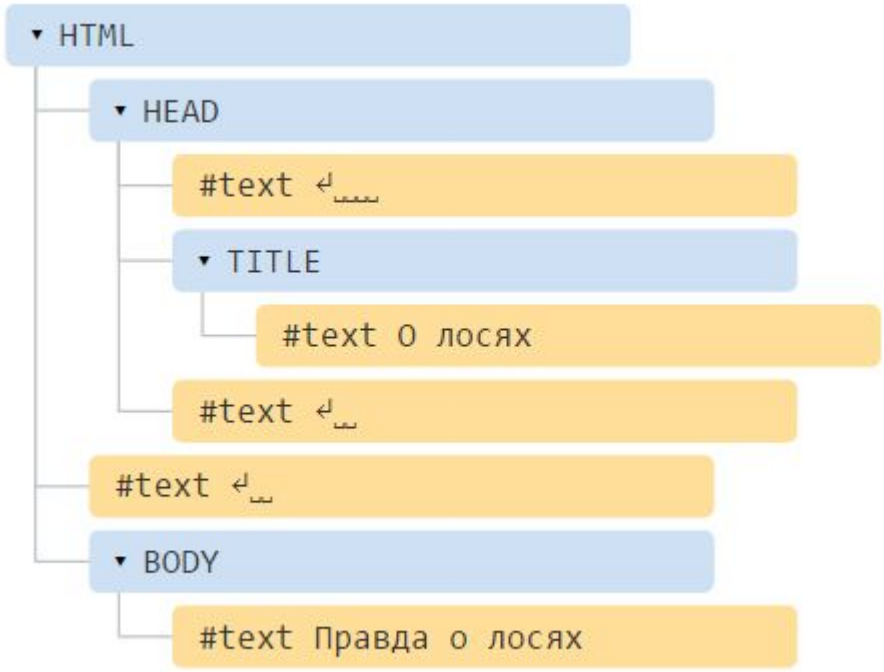
Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

Дерево DOM

```
<!DOCTYPE HTML>
<html>
<head>
  <title>О лосях</title>
</head>
<body>
  Правда о лосях
</body>
</html>
```



Дерево DOM

В вышеприведенном примере DOM-дереве выделено два типа узлов.

1. Теги образуют узлы-элементы (element node). Естественным образом одни узлы вложены в другие. Структура дерева образована исключительно за счет них.
2. Текст внутри элементов образует текстовые узлы (text node), обозначенные как #text. Текстовый узел содержит исключительно строку текста и не может иметь потомков, то есть он всегда на самом нижнем уровне.

Автоисправление

При чтении неверного HTML браузер автоматически корректирует его для показа и при построении DOM.

В частности, всегда будет верхний тег <html>. Даже если в тексте нет – в DOM он будет, браузер создаст его самостоятельно.

То же самое касается и тега <body>.

Например, если файл состоит из одного слова "Привет", то браузер автоматически обернёт его в <html> и <body>.

Навигация по DOM-элементам

Более правильной и общепринятой практикой является доступ к элементу вызовом **document.getElementById**("идентификатор").

```
<div id="content">Выделим этот элемент</div>

<script>
  var elem = document.getElementById('content');

  elem.style.background = 'red';

</script>
```

Навигация по DOM-элементам

getElementsByTagName

Метод elem.getElementsByTagName(tag) ищет все элементы с заданным тегом tag внутри элемента elem и возвращает их в виде списка.

Регистр тега не имеет значения.

```
// получить все div-элементы
var elements = document.getElementsByTagName('div');
```

в отличие от **getElementById**, который существует только в контексте document, метод **getElementsByTagName** может искать внутри любого элемента.

Навигация по DOM-элементам

document.getElementsByName

Вызов document.getElementsByName(name) позволяет получить все элементы с данным атрибутом name.

getElementsByClassName

Вызов elem.getElementsByClassName(className) возвращает коллекцию элементов с классом className. Находит элемент и в том случае, если у него несколько классов, а искомый – один из них.

Навигация по DOM-элементам

querySelectorAll

Вызов elem.querySelectorAll(css) возвращает все элементы внутри elem, удовлетворяющие CSS-селектору css.

Это один из самых часто используемых и полезных методов при работе с DOM.

querySelector

Вызов elem.querySelector(css) возвращает не все, а только первый элемент, соответствующий CSS-селектору css.

Иначе говоря, результат – такой же, как и при elem.querySelectorAll(css)[0], но в последнем вызове сначала ищутся все элементы, а потом берётся первый, а в elem.querySelector(css) ищется только первый, то есть он эффективнее.

Этот метод часто используется, когда мы заведомо знаем, что подходящий элемент только один, и хотим получить в переменную сразу его.

Навигация по DOM-элементам

Метод	Ищет по...	Ищет внутри элемента?	Поддержка
`getElementById`	`id`	-	везде
`getElementsByName`	`name`	-	везде
`getElementsByTagName`	тег или `'*'`	✓	везде
`getElementsByClassName`	классу	✓	кроме IE8-
`querySelector`	CSS-селектор	✓	везде
`querySelectorAll`	CSS-селектор	✓	везде

Объектная модель браузера (BOM)

BOM – это объекты для работы с чем угодно, кроме документа.

- Объект navigator содержит общую информацию о браузере и операционной системе. Особенно примечательны два свойства: navigator.userAgent – содержит информацию о браузере и navigator.platform – содержит информацию о платформе, позволяет различать Windows/Linux/Mac и т.п.
- Объект location содержит информацию о текущем URL страницы и позволяет перенаправить посетителя на новый URL.
- Функции alert/confirm/prompt – тоже входят в BOM.

Браузерные события

Для реакции на действия посетителя и внутреннего взаимодействия скриптов существуют *события*.

Событие – это сигнал от браузера о том, что что-то произошло. Существует много видов событий.

События мыши:

- click – происходит, когда кликнули на элемент левой кнопкой мыши
- contextmenu – происходит, когда кликнули на элемент правой кнопкой мыши
- mouseover – возникает, когда на элемент наводится мышь
- mousedown и mouseup – когда кнопку мыши нажали или отжали
- mousemove – при движении мыши

Браузерные события

События на элементах управления:

- submit – посетитель отправил форму <form>
- focus – посетитель фокусируется на элементе, например нажимает на <input>
- blur – потеря фокуса

Клавиатурные события:

- keydown – когда посетитель нажимает клавишу
- keyup – когда посетитель отпускает клавишу

События документа:

- DOMContentLoaded – когда HTML загружен и обработан, DOM документа полностью построен и доступен.

События CSS:

- transitionend – когда CSS-анимация завершена.

Назначение обработчиков событий

Событию можно назначить *обработчик*, то есть функцию, которая сработает, как только событие произошло.

Именно благодаря обработчикам JavaScript-код может реагировать на действия посетителя.

Есть несколько способов назначить событию обработчик. Ниже мы их рассмотрим, начиная от самого простого.

Назначение обработчиков событий

Использование атрибута HTML

Обработчик может быть назначен прямо в разметке, в атрибуте, который называется on<событие>.

Например, чтобы прикрепить click-событие к input кнопке, можно присвоить обработчик onclick, вот так:

```
<input value="Нажми меня" onclick="alert('Клик!')" type="button">
```

Так как DOM-свойство onclick, в итоге, одно, то назначить более одного обработчика так нельзя.

Объект события

Чтобы хорошо обработать событие, недостаточно знать о том, что это – «клик» или «нажатие клавиши». Могут понадобиться детали: координаты курсора, введённый символ и другие, в зависимости от события.

Детали произошедшего браузер записывает в «объект события», который передаётся первым аргументом в обработчик.

Свойства объекта event

- **event.type -** Тип события
- **event.currentTarget -** Элемент, на котором сработал обработчик.
- **event.clientX / event.clientY -** Координаты курсора

Есть также и ряд других свойств, в зависимости от событий

Всплытие и перехват

Основной принцип всплытия:

При наступлении события обработчики сначала срабатывают на самом вложенном элементе, затем на его родителе, затем выше и так далее, вверх по цепочке вложенности.

Прекращение всплытия

Любой промежуточный обработчик может решить, что событие полностью обработано, и остановить всплытие.

Для остановки всплытия нужно вызвать метод event.stopPropagation().

Целевой элемент event.target

На каком бы элементе мы ни поймали событие, всегда можно узнать, где конкретно оно произошло.

Самый глубокий элемент, который вызывает событие, называется «целевым» или «исходным» элементом и доступен как event.target.

Делегирование событий

Всплытие событий позволяет реализовать один из самых важных приёмов разработки – делегирование.

Он заключается в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому – мы ставим один обработчик на их общего предка. Из него можно получить целевой элемент event.target, понять на каком именно потомке произошло событие и обработать его.