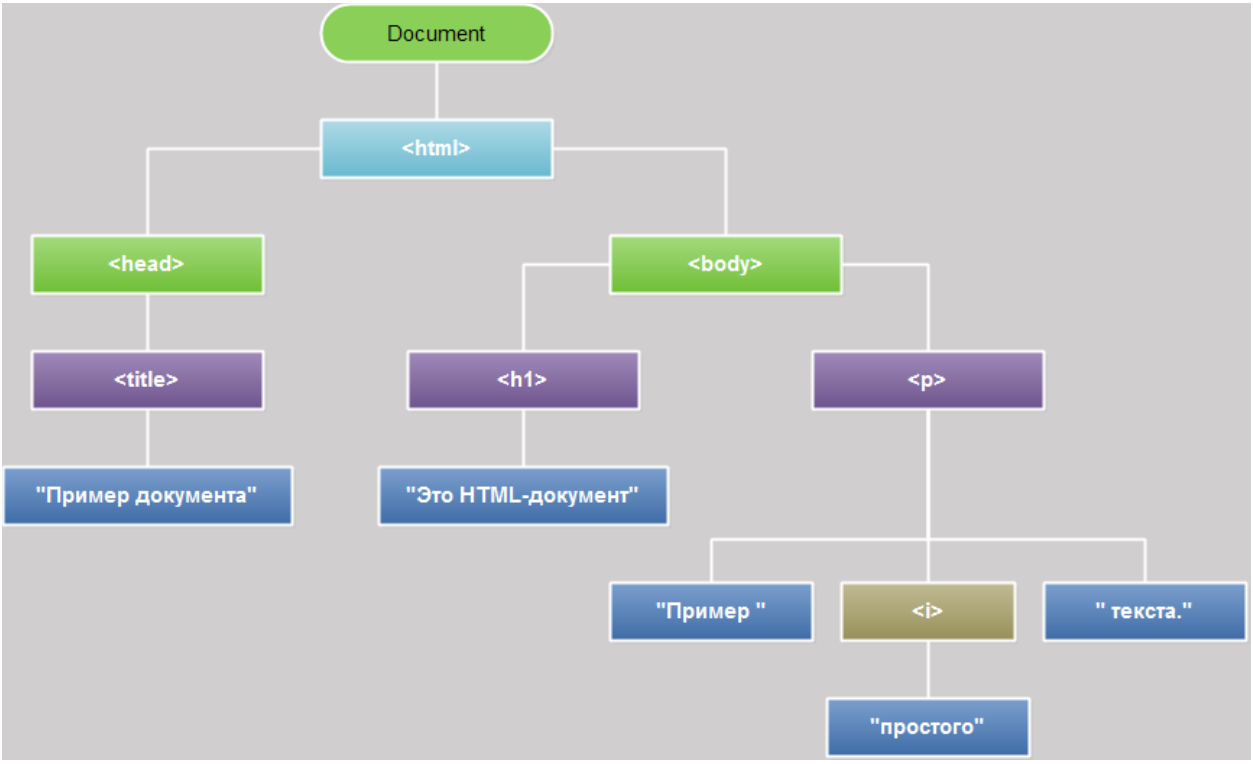
Обзор объектной модели W3C DOM

Рассмотрев раннюю упрощенную модель DOM, теперь обратимся к более мощной и стандартизованной модели W3C DOM, пришедшей ей на смену. Программный интерфейс (API) модели W3C DOM не особенно сложен, но прежде чем перейти к рассмотрению DOMпрограммирования, необходимо уточнить несколько вещей относительно DOM архитектуры.

Представление документов в виде дерева

HTML-документы имеют иерархическую структуру вложенных тегов, которая в DOM представлена в виде дерева объектов. Узлы дерева представляют различные типы содержимого документа. В первую очередь, древовидное представление HTML-документа содержит узлы, представляющие элементы или теги, такие как <body> и <p>, и узлы, представляющие строки текста. HTML-документ также может содержать узлы, представляющие HTML-комментарии. Рассмотрим следующий простой HTML-документ:

```
<html>
	<head>
		<title>Пример документа</title>
	</head>
	<body>
		<h1>Это HTML-документ</h1>
		<p>Пример <i>простого</i> текста.</p>
	</body>
</html>
```



Узел, расположенный непосредственно над данным узлом, называется родительским по отношению к данному узлу. Узлы, расположенные на один уровень ниже другого узла, являются дочерними по отношению к данному узлу. Узлы, находящиеся на том же уровне и имеющие того

же родителя, называются братьями (siblings). Узлы, расположенные на любое число уровней ниже другого узла, являются его потомками. Родительские, прародительские и любые другие узлы, расположенные выше данного узла, являются его предками.

Каждый прямоугольник на этой диаграмме является узлом документа, который представлен объектом Node. Обратите внимание, что на рисунке изображено три различных типа узлов. Корнем дерева является узел Document, который представляет документ целиком. Узлы, представляющие HTML-элементы, являются узлами типа Element, а узлы, представляющие текст, - узлами типа Text. Document, Element и Text - это подклассы класса Node. Document и Element являются двумя самыми важными классами в модели DOM.

Интерфейс Node определяет свойства и методы для перемещения по дереву и манипуляций им. Свойство childNodes объекта Node возвращает список дочерних узлов, свойства firstChild, lastChild, nextSibling, previousSibling и parentNode предоставляют средство обхода узлов дерева.

Такие методы, как appendChild(), removeChild(), replaceChild() и insertBefore(), позволяют добавлять узлы в дерево документа и удалять их.

Типы узлов

Типы узлов в дереве документа представлены специальными подынтерфейсами интерфейса Node. У любого объекта Node есть свойство nodeType, определяющее тип данного узла. Если свойство nodeType узла равно, например, константе Node.ELEMENT_NODE, значит, объект Node является также объектом Element, и можно использовать с ним все методы и свойства, определенные интерфейсом Element. В табл. 15.1 перечислены чаще всего встречающиеся в HTMLдокументах типы узлов и значения nodeType для каждого из них.

Интерфейс Константа nodeType Значение nodeType

Element Node.ELEMENT_NODE 1

Text Node.TEXT_NODE 3

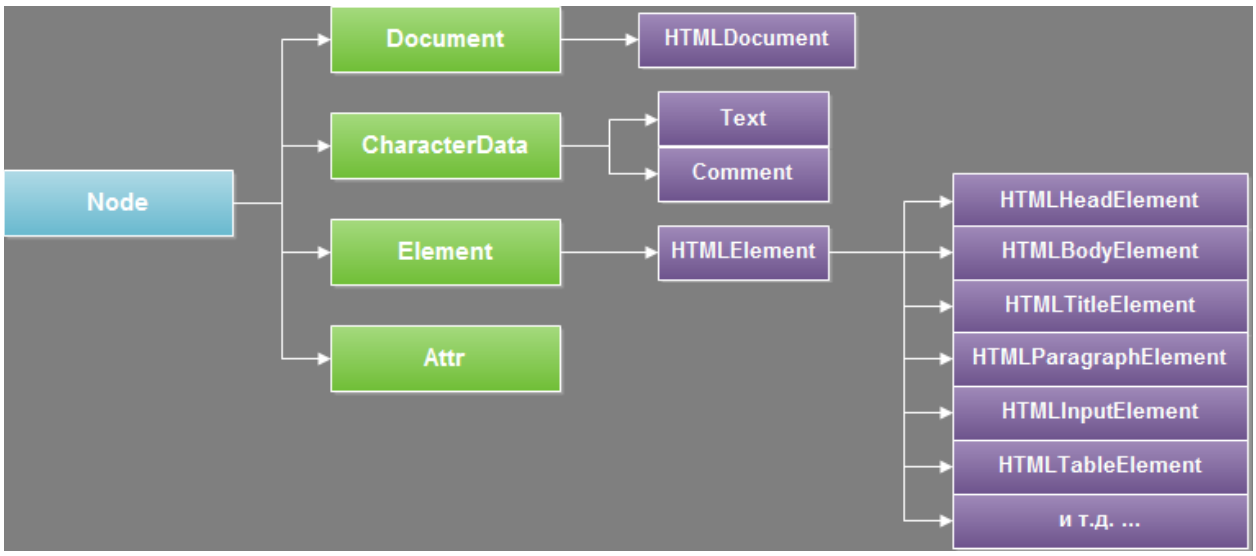
Document Node.DOCUMENT_NODE 9

Comment Node.COMMENT_NODE 8

DocumentFragment Node.DOCUMENT_FRAGMENT_NODE 11

Attr Node.ATTRIBUTE_NODE 2

Тип Node и его подтипы образуют иерархию типов, изображенную на диаграмме ниже. Обратите внимание на формальные отличия между обобщенными типами Document и Element, и типами HTMLDocument и HTMLElement. Тип Document представляет HTML и XML-документ, а класс Element представляет элемент этого документа. Подклассы HTMLDocument и HTMLElement представляют конкретно HTML-документ и его элементы:



Иерархия классов узлов JavaScript

На этой диаграмме следует также отметить наличие большого количества подтипов класса HTMLElement, представляющих конкретные типы HTML-элементов. Каждый из них определяет JavaScript-свойства, отражающие HTML-атрибуты конкретного элемента или группы элементов. Некоторые из этих специфических классов определяют дополнительные свойства или методы, которые не являются отражением синтаксиса языка разметки HTML.

Выбор элементов документа

Работа большинства клиентских программ на языке JavaScript так или иначе связана с манипулированием элементами документа. В ходе выполнения эти программы могут использовать глобальную переменную document, ссылающуюся на объект Document. Однако, чтобы выполнить какие-либо манипуляции с элементами документа, программа должна каким-то образом получить, или выбрать, объекты Element, ссылающиеся на эти элементы документа. Модель DOM определяет несколько способов выборки элементов. Выбрать элемент или элементы документа можно:

- по значению атрибута id;
- по значению атрибута name;
- по имени тега;
- по имени класса или классов CSS;
- по совпадению с определенным селектором CSS.

Выбор элементов по значению атрибута id

Все HTML-элементы имеют атрибуты id. Значение этого атрибута должно быть уникальным в пределах документа - никакие два элемента в одном и том же документе не должны иметь одинаковые значения атрибута id. Выбрать элемент по уникальному значению атрибута id можно с помощью метода getElementById() объекта Document:

```
var section1 = document.getElementById("section1");
```

Это самый простой и самый распространенный способ выборки элементов. Если сценарию необходимо иметь возможность манипулировать каким-то определенным множеством элементов документа, присвойте значения атрибутам id этих элементов и используйте возможность их поиска по этим значениям.

В версиях Internet Explorer ниже IE8 метод getElementById() выполняет поиск значений атрибутов id без учета регистра символов и, кроме того, возвращает элементы, в которых будет найдено совпадение со значением атрибута name.

Выбор элементов по значению атрибута name

HTML-атрибут name первоначально предназначался для присваивания имен элементам форм, и значение этого атрибута использовалось, когда выполнялась отправка данных формы на сервер. Подобно атрибуту id, атрибут name присваивает имя элементу. Однако, в отличие от id, значение атрибута name не обязано быть уникальным: одно и то же имя могут иметь сразу несколько элементов, что вполне обычно при использовании в формах радиокнопок и флажков. Кроме того, в отличие от id, атрибут name допускается указывать лишь в некоторых HTML-элементах, включая формы, элементы форм и элементы <iframe> и <img>.

Выбрать HTML-элементы, опираясь на значения их атрибутов name, можно с помощью метода getElementsByName() объекта Document:

```
var radiobuttons = document.getElementsByName("favorite_color");
```

Метод getElementsByName() определяется не классом Document, а классом HTMLDocument, поэтому он доступен только в HTML-документах и не доступен в XML-документах. Он возвращает объект NodeList, который ведет себя, как доступный только для чтения массив объектов Element.

В IE метод getElementsByName() возвращает также элементы, значения атрибутов id которых совпадает с указанным значением. Чтобы обеспечить совместимость с разными версиями браузеров, необходимо внимательно подходить к выбору значений атрибутов и не использовать одни и те же строки в качестве значений атрибутов name и id.

Выбор элементов по типу

Метод getElementsByTagName() объекта Document позволяет выбрать все HTML или XML-элементы указанного типа (или по имени тега). Например, получить подобный массиву объект, доступный только для чтения, содержащий объекты Element всех элементов в документе, можно следующим образом:

```
var spans = document.getElementsByTagName("span");
```

Подобно методу getElementsByName(), getElementsByTagName() возвращает объект NodeList. Элементы документа включаются в массив NodeList в том же порядке, в каком они следуют в документе, т.е. первый элемент <p> в документе можно выбрать так:

```
var firstParagraph = document.getElementsByTagName("p")[0];
```

Имена HTML-тегов не чувствительны к регистру символов, и когда getElementsByTagName() применяется к HTML-документу, он выполняет сравнение с именем тега

без учета регистра символов. Переменная spans, созданная выше, например, будет включать также все элементы , которые записаны как <SPAN>.

Можно получить NodeList, содержащий все элементы документа, если передать методу getElementsByTagName() шаблонный символ «*».

Кроме того, классом Element также определяет метод getElementsByTagName(). Он действует точно так же, как и версия метода в классе Document, но выбирает только элементы, являющиеся потомками для элемента, относительно которого вызывается метод. То есть отыскать все элементы внутри первого элемента <p> можно следующим образом:

```
var firstParagraph = document.getElementsByTagName("p")[0];

var firstParagraphSpans = firstParagraph.getElenentsByTagName("span");
```

Выбор элементов по классу CSS

Значением HTML-атрибута class является список из нуля или более идентификаторов, разделенных пробелами. Он дает возможность определять множества связанных элементов документа: любые элементы, имеющие в атрибуте class один и тот же идентификатор, являются частью одного множества. Слово class зарезервировано в языке JavaScript, поэтому для хранения значения HTML-атрибута class в клиентском JavaScript используется свойство className.

Обычно атрибут class используется вместе с каскадными таблицами стилей CSS, с целью применить общий стиль отображения ко всем членам множества. Однако кроме этого, стандарт HTML5 определяет метод getElementsByClassName(), позволяющий выбирать множества элементов документа на основе идентификаторов в их атрибутах class.

Подобно методу getElementsByTagName(), метод getElementsByClassName() может вызываться и для HTML-документов, и для HTML-элементов, и возвращает «живой» объект NodeList, содержащий все потомки документа или элемента, соответствующие критерию поиска.

Метод getElementsByClassName() принимает единственный строковый аргумент, но в самой строке может быть указано несколько идентификаторов, разделенных пробелами. Соответствующими будут считаться все элементы, атрибуты class которых содержат все указанные идентификаторы. Порядок следования идентификаторов не имеет значения. Обратите внимание, что и в атрибуте class, и в аргументе метода getElementsByClassName() идентификаторы классов разделяются пробелами, а не запятыми.

Ниже приводится несколько примеров использования метода getElementsByClassName():

```
// Отыскать все элементы с классом "warning"
var warnings = document.getElementsByClassName("warning");

// Отыскать всех потомков элемента с идентификаторам "log"
// с классами "error" и "fatal"
var log = document.getElementById("log");
var fatal = log.getElementsByClassName("fatal error");
```

Выбор элементов с использованием селекторов CSS

Каскадные таблицы стилей CSS имеют очень мощные синтаксические конструкции, известные как селекторы, позволяющие описывать элементы или множества элементов документа. Наряду со стандартизацией селекторов CSS3, другой стандарт консорциума W3C, известный как Selectors API, определяет методы JavaScript для получения элементов, соответствующих указанному селектору.

Ключевым в этом API является метод querySelectorAll() объекта Document. Он принимает единственный строковый аргумент с селектором CSS и возвращает объект NodeList, представляющий все элементы документа, соответствующие селектору.

В дополнение к методу querySelectorAll() объект документа также определяет метод querySelector(), подобный методу querySelectorAll(), - с тем отличием, что он возвращает только первый (в порядке следования в документе) соответствующий элемент или null, в случае отсутствия соответствующих элементов.

Эти два метода также определяются классом Elements. Когда они вызываются относительно элемента, поиск соответствия заданному селектору выполняется во всем документе, а затем результат фильтруется так, чтобы в нем остались только потомки использованного элемента. Такой подход может показаться противоречащим здравому смыслу, так как он означает, что строка селектора может включать предков элемента, для которого выполняется сопоставление.

Добавление и удаление узлов

Давайте рассмотрим, как создавать новые элементы «на лету» и заполнять их данными.В качестве примера рассмотрим добавление сообщения на страницу, чтобы оно было оформлено красивее чем обычный alert.

```
HTML-код для сообщения:
<div class="alert">
  <strong>Ура!</strong> Вы прочитали это важное сообщение.
</div>
```

Создание элемента

Для создания элементов используются следующие методы:

document.createElement(tag)
Создает новый элемент с указанным тегом:
var div = document.createElement('div');

document.createTextNode(text)
Создает новый *текстовый* узел с данным текстом:
var textElem = document.createTextNode('Тут был я');

В нашем случае мы хотим сделать DOM-элемент div, дать ему классы и заполнить текстом:

```
var div = document.createElement('div');
div.className = "alert alert-success";
div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";
```

После этого кода у нас есть готовый DOM-элемент. Пока что он присвоен в переменную div, но не виден, так как никак не связан со страницей.

Добавление элемента: appendChild, insertBefore

Чтобы DOM-узел был показан на странице, его необходимо вставить в document.

Для этого первым делом нужно решить, куда мы будем его вставлять. Предположим, что мы решили, что вставлять будем в некий элемент parentElem, например var parentElem = document.body.

Для вставки внутрь parentElem есть следующие методы:

parentElem.appendChild(elem)

Добавляет elem в конец дочерних элементов parentElem.

Следующий пример добавляет новый элемент в конец <ol>:

```
 <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>
```

parentElem.insertBefore(elem, element)

Вставляет elem в коллекцию детей parentElem, перед элементом element.

Следующий код вставляет новый элемент перед вторым <li>:

```
<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>
```

Для вставки элемента в начало достаточно указать, что вставлять будем перед первым потомком:

```
list.insertBefore(newLi, list.firstChild);
```

У читателя, который посмотрит на этот код внимательно, наверняка возникнет вопрос: «А что, если list вообще пустой, в этом случае ведь list.firstChild = null, произойдёт ли вставка?»

Ответ – да, произойдёт.

Дело в том, что если вторым аргументом указать null, то insertBefore сработает как appendChild:

```
parentElem.insertBefore(elem, null);

// то же, что и:

parentElem.appendChild(elem)
```

Так что insertBefore универсален.

На заметку:

Все методы вставки возвращают вставленный узел.

Примеры использования:

Добавим сообщение в конец <body>:

```
<script>
  var div = document.createElement('div');
  div.className = "alert alert-success";
  div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";

  document.body.appendChild(div);
</script>
```

…А теперь – в начало <body>:

```
<script>
  var div = document.createElement('div');
  div.className = "alert alert-success";
  div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";

  document.body.insertBefore(div, document.body.firstChild);
</script>
```

Клонирование узлов: cloneNode

А как бы вставить второе похожее сообщение? Конечно, можно сделать функцию для генерации сообщений и поместить туда этот код, но в ряде случаев гораздо эффективнее –

клонировать существующий div, а потом изменить текст внутри. В частности, если элемент большой, то клонировать его будет гораздо быстрее, чем пересоздавать.

Вызов elem.cloneNode(true) создаст «глубокую» копию элемента – вместе с атрибутами, включая подэлементы. Если же вызвать с аргументом false, то копия будет сделана без дочерних элементов. Это нужно гораздо реже.

Удаление узлов: removeChild

Для удаления узла есть два метода:

parentElem.removeChild(elem)

Удаляет elem из списка детей parentElem.

parentElem.replaceChild(newElem, elem)

Среди детей parentElem удаляет elem и вставляет на его место newElem.

Оба этих метода возвращают удаленный узел, то есть elem. Если нужно, его можно вставить в другое место DOM тут же или в будущем.

На заметку:

Если вы хотите переместить элемент на новое место – не нужно его удалять со старого.

Все методы вставки автоматически удаляют вставляемый элемент со старого места.

Например, поменяем элементы местами:

```
 <div>Первый</div>
<div>Второй</div>
<script>
  var first = document.body.children[0];
  var last = document.body.children[1];

  // нет необходимости в предварительном removeChild(last)
  document.body.insertBefore(last, first); // поменять местами
</script>
```

Метод remove

В современном стандарте есть также метод elem.remove(), который удаляет элемент напрямую, не требуя ссылки на родителя. Это зачастую удобнее, чем removeChild.

Удаление сообщения

Сделаем так, что через секунду сообщение пропадёт:

```
<script>
  var div = document.createElement('div');
  div.className = "alert alert-success";
  div.innerHTML = "<strong>Ура!</strong> Вы прочитали это важное сообщение.";

  document.body.appendChild(div);

  setTimeout(function() {
    div.parentNode.removeChild(div);
  }, 1000);
</script>
```

Текстовые узлы для вставки текста

При работе с сообщением мы использовали только узлы-элементы и innerHTML. Но и текстовые узлы тоже имеют интересную область применения, если текст для сообщения нужно показать именно как текст, а не как HTML, то можно обернуть его в текстовый узел.

Например:

```
<script>
  var div = document.createElement('div');
  div.className = "alert alert-success";
  document.body.appendChild(div);

  var text = prompt("Введите текст для сообщения", "Жили были <a> и <b>!");

  // вставится именно как текст, без HTML-обработки
  div.appendChild(document.createTextNode(text));
</script>
```

Итого

Методы для создания узлов:

document.createElement(tag) – создает элемент

document.createTextNode(value) – создает текстовый узел

elem.cloneNode(deep) – клонирует элемент, если deep == true, то со всеми потомками, если false – без потомков.

Вставка и удаление узлов:

parent.appendChild(elem)

parent.insertBefore(elem, nextSibling)

parent.removeChild(elem)

parent.replaceChild(newElem, elem)

Все эти методы возвращают elem.

Практическое задание

Реализуйте динамическое изменение дерева узлов вашей странички. Изменение должно инициироваться событиями. Для регистрации событий используйте метод addEventListener, для выборки элементов привязки используйте различные способы.

Примеры работ: добавление и удаление элементов списка, реализация «однооконного» приложения, «карусель» картинок, раскрытие картинки из превью на полный экран и т.д.