12 DOM Level 2 и 3

- Изменения в спецификациях Level 2 и 3
- DOM API для работы со стилями
- Обход и диапазоны DOM

Базовая структура HTML- и XML-документов, определенная в спецификации DOM Level 1, была расширена в DOM Level 2 и 3 интерактивными возможностями и улучшенными XML-механизмами. В результате на данный момент спецификации DOM Level 2 и 3 составляют следующие связанные модули, описывающие конкретные подмножества DOM:

- **DOM Core** — дополняет DOM Level 1 Core, добавляя к узлам методы и свойства;
- **DOM Views** — определяет для документа разные представления на основе стилей;
- **DOM Events** — обеспечивает интерактивность DOM-документов с помощью событий;
- **DOM Style** — описывает программный доступ к CSS-стилям и их изменение;
- **DOM Traversal and Range** — предоставляет новые интерфейсы для обхода DOM-документа и выделения его частей;
- **DOM HTML** — расширяет HTML Level 1 новыми свойствами, методами и интерфейсами.

В данной главе мы обсудим все эти модули, кроме событий DOM, которые подробно описаны в главе 13.

DOM Level 3 содержит также модули XPath и Load and Save, описываемые в главе 18.

Изменения DOM

Спецификации DOM Level 2 и 3 Core были разработаны для того, чтобы реализовать в DOM API все требования языка XML и улучшить обработку ошибок и распознавание функциональных возможностей. В основном это сводится к поддержке XML-пространств имен. DOM Level 2 Core не определяет никаких новых типов, а просто добавляет новые методы и свойства к типам DOM Level 1. DOM Level 3 Core расширяет доступные типы и определяет несколько новых.

Модули DOM Views и DOM HTML, которые также содержат ряд новых свойств и методов, довольно малы, и поэтому мы обсудим их вместе с DOM Core. Узнать, поддерживает ли браузер эти части DOM, можно следующим образом:

```
var supportsDOM2Core = document.implementation.hasFeature("Core", "2.0");
var supportsDOM3Core = document.implementation.hasFeature("Core", "3.0");
var supportsDOM2HTML = document.implementation.hasFeature("HTML", "2.0");
var supportsDOM2Views = document.implementation.hasFeature("Views", "2.0");
var supportsDOM2XML = document.implementation.hasFeature("XML", "2.0");
```

В этой главе описаны только те части DOM, которые уже реализованы в браузерах.

XML-пространства имен

XML-пространства имен позволяют использовать элементы из разных XML-подобных языков в одном документе правильного формата без риска конфликтов имен. Технически XML-пространства имен не поддерживаются в HTML, поэтому примеры в этом разделе написаны на XHTML.

Пространства имен указываются с помощью атрибута `xmlns`. Языку XHTML соответствует пространство имен `http://www.w3.org/1999/xhtml`, которое нужно включать в элемент `<html>` любой XHTML-страницы правильного формата, например:

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Example XHTML page</title>
    </head>
    <body>
        Hello world!
    </body>
</html>
```

В этом примере все элементы считаются по умолчанию частью пространства имен XHTML. Можно явно создать префикс для пространства имен XML, указав атрибут `xmlns`, двоеточие и префикс, например:

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <xhtml:head>
```

```
        <xhtml:title>Example XHTML page</xhtml:title>
    </xhtml:head>
    <xhtml:body>
        Hello world!
    </xhtml:body>
</xhtml:html>
```

Здесь определяется пространство имен XHTML с префиксом `xhtml`, в результате все XHTML-элементы должны начинаться с этого префикса. Пространства имен можно также использовать с атрибутами для предотвращения конфликтов между языками:

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <xhtml:head>
        <xhtml:title>Example XHTML page</xhtml:title>
    </xhtml:head>
    <xhtml:body xhtml:class="home">
        Hello world!
    </xhtml:body>
</xhtml:html>
```

Атрибуту `class` в этом примере предшествует префикс `xhtml`. Указывать пространства имен не требуется, если в документе используется только один язык, основанный на XML, но это полезно, если языков больше. Например, следующий документ содержит XHTML- и SVG-код:

```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Example XHTML page</title>
    </head>
    <body>
        <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
             viewBox="0 0 100 100" style="width:100%; height:100%">
            <rect x="0" y="0" width="100" height="100" style="fill:red" />
        </svg>
    </body>
</html>
```

В этом примере для элемента `<svg>` задано отдельное пространство имен `http://www.w3.org/2000/svg`, которое указывает, что он является посторонним для документа. Оно применяется также ко всем дочерним элементам элемента `<svg>` и ко всем их атрибутам. Хотя технически это XHTML-документ, благодаря пространству имен SVG-код обрабатывается правильно.

При вызове методов, работающих с узлами такого документа, возникают интересные проблемы. Например, к какому пространству имен будут относиться создаваемые элементы? Какие элементы будут возвращены, если запросить теги с конкретным именем? К счастью, в DOM Level 2 Core для большинства методов DOM Level 1 определены версии, учитывающие пространства имен.

Изменения типа Node

В DOM Level 2 тип `Node` включает следующие новые свойства, учитывающие пространства имен:

- `localName` — имя узла без префикса пространства имен;
- `namespaceURI` — URI пространства имен узла или `null`, если свойство не задано;
- `prefix` — префикс пространства имен или `null`, если свойство не задано.

Если для узла задан префикс пространства имен, свойство `nodeName` эквивалентно значению `prefix + ":" + localName`. Рассмотрим пример:

Листинг NamespaceExample.xml


```
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Example XHTML page</title>
    </head>
    <body>
        <s:svg xmlns:s="http://www.w3.org/2000/svg" version="1.1"
             viewBox="0 0 100 100" style="width:100%; height:100%">
            <s:rect x="0" y="0" width="100" height="100" style="fill:red" />
        </s:svg>
    </body>
</html>
```

У элемента `<html>` свойства `localName` и `tagName` имеют значение `"html"`, `namespaceURI` — значение `"http://www.w3.org/1999/xhtml"`, а `prefix` — значение `null`. У элемента `<s:svg>` свойство `localName` имеет значение `"svg"`, `tagName` — значение `"s:svg"`, `namespaceURI` — значение `"http://www.w3.org/2000/svg"`, а `prefix` — значение `"s"`.

В DOM Level 3 представлены следующие методы для работы с пространствами имен:

- `isDefaultNamespace(`*URIПространстваИмен*`)` — возвращает `true`, если аргумент представляет пространство имен, предлагаемое по умолчанию для узла;
- `lookupNamespaceURI(`*префикс*`)` — возвращает URI пространства имен для указанного префикса;
- `lookupPrefix(`*URIПространстваИмен*`)` — возвращает префикс для указанного URI пространства имен.

Для предыдущего примера эти методы возвратили бы следующие значения:

```
// возвращает true
alert(document.body.isDefaultNamespace("http://www.w3.org/1999/xhtml"));

// предполагается, что svg содержит ссылку на элемент <s:svg>
alert(svg.lookupPrefix("http://www.w3.org/2000/svg"));    // "s"
alert(svg.lookupNamespaceURI("s"));    // "http://www.w3.org/2000/svg"
```

Эти методы полезны, если имеется ссылка на узел и нужно определить, как он связан с остальным документом.

Изменения типа Document

В DOM Level 2 тип `Document` содержит следующие новые методы, специфичные для пространств имен:

- `createElementNS(`*URIПространстваИмен*, *имяТега*`)` — создает элемент с заданным именем тега в пространстве имен с указанным URI;
- `createAttributeNS(`*URIПространстваИмен*, *имяАтрибута*`)` — создает узел атрибута в пространстве имен с указанным URI;
- `getElementsByTagNameNS(`*URIПространстваИмен*, *имяТега*`)` — возвращает коллекцию `NodeList`, содержащую элементы с заданным именем тега из пространства имен с указанным URI.

Обратите внимание, что в эти методы передается URI нужного пространства имен (а не префикс), например:

```
// создание SVG-элемента
var svg = document.createElementNS("http://www.w3.org/2000/svg", "svg");

// создание атрибута random в пространстве имен
var att = document.createAttributeNS("http://www.somewhere.com", "random");

// получение всех XHTML-элементов
var elems =
    document.getElementsByTagNameNS("http://www.w3.org/1999/xhtml", "*");
```

Методы, специфичные для пространств имен, могут потребоваться, только если документ содержит более одного пространства имен.

Изменения типа Element

Изменения типа `Element` в DOM Level 2 Core связаны в основном с атрибутами. Он содержит следующие новые методы:

- `getAttributeNS(`*URIПространстваИмен*, *локальноеИмя*`)` — получает атрибут с заданным именем из пространства имен с указанным URI;
- `getAttributeNodeNS(`*URIПространстваИмен*, *локальноеИмя*`)` — получает узел атрибута с заданным именем из пространства имен с указанным URI;
- `getElementsByTagNameNS(`*URIПространстваИмен*, *имяТега*`)` — возвращает коллекцию `NodeList`, содержащую элементы из поддерева опорного элемента, которые имеют заданное имя тега и относятся к пространству имен с указанным URI;
- `hasAttributeNS(`*URIПространстваИмен*, *локальноеИмя*`)` — определяет, есть ли у элемента атрибут с заданным именем из пространства имен с указанным URI

(в DOM Level 2 Core есть также метод `hasAttribute()`, не учитывающий пространство имен);

- `removeAttributeNS(`*URIПространстваИмен*, *локальноеИмя*`)` — удаляет атрибут с заданным именем из пространства имен с указанным URI;
- `setAttributeNS(`*URIПространстваИмен*, *квалифицированноеИмя*, *значение*`)` — присваивает указанное значение атрибуту с заданным именем из пространства имен с указанным URI;
- `setAttributeNodeNS(`*узелАтрибута*`)` — задает узел атрибута из пространства имен с указанным URI.

Эти методы работают так же, как их аналоги из DOM Level 1, и отличаются от них только первым аргументом, которым у всех методов, кроме `setAttributeNodeNS()`, является URI пространства имен.

Изменения типа NamedNodeMap

Тип `NamedNodeMap` также содержит несколько новых методов для работы с пространствами имен. Поскольку он используется для представления атрибутов, эти методы в основном применяются к атрибутам:

- `getNamedItemNS(`*URIПространстваИмен*, *локальноеИмя*`)` — получает элемент с заданным именем из пространства имен с указанным URI;
- `removeNamedItemNS(`*URIПространстваИмен*, *локальноеИмя*`)` — удаляет элемент с заданным именем из пространства имен с указанным URI;
- `setNamedItemNS(`*узел*`)` — добавляет узел, которому уже должно быть назначено пространство имен.

Эти методы используются редко, потому что доступ к атрибутам обычно осуществляется через элемент.

Другие изменения

DOM Level 2 Core содержит также ряд других небольших изменений DOM, которые не имеют отношения к XML-пространствам имен и были внесены в основном для обеспечения гибкости и полноты API.

Изменения типа DocumentType

В тип `DocumentType` добавлены свойства `publicId`, `systemId` и `internalSubset`. Первые два из них представляют данные, которые содержатся в объявлении типа документа, но недоступны в DOM Level 1. Рассмотрим следующее объявление HTML-документа:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
```

Здесь свойство `publicId` имеет значение `"-//W3C//DTD HTML 4.01//EN"`, а `systemId` — значение `"http://www.w3.org/TR/html4/strict.dtd"`. Если браузер поддерживает DOM Level 2, он должен быть способен выполнить следующий код:

```
alert(document.doctype.publicId);
alert(document.doctype.systemId);
```

Едва ли эти сведения когда-нибудь могут потребоваться на веб-страницах.

Свойство `internalSubset` обеспечивает доступ к любым дополнительным определениям в объявлении типа документа, например:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
[<!ELEMENT name (#PCDATA)>] >
```

Для этого кода свойство `document.doctype.internalSubset` возвращает `"<!ELEMENT name (#PCDATA)>"`. Оно редко используется в HTML и чуть чаще в XML.

Изменения типа Document

Единственный новый метод типа `Document`, не связанный с пространствами имен, — это метод `importNode()`, который импортирует узел из одного документа в другой, чтобы его можно было добавить в структуру документа. Если помните, у каждого узла есть свойство `ownerDocument`, которое указывает, к какому документу относится узел. Если в метод вроде `appendChild()` передается узел, у которого свойство `ownerDocument` указывает на другой документ, происходит ошибка. При вызове метода `importNode()` для узла из другого документа возвращается новая версия узла, принадлежащая текущему документу.

Метод `importNode()` похож на метод `cloneNode()` элемента. Он принимает узел, который нужно клонировать, и логическое значение, указывающее, нужно ли также скопировать его дочерние узлы, и возвращает копию узла, например:

```
// импорт узла и всех его дочерних узлов
var newNode = document.importNode(oldNode, true);
document.body.appendChild(newNode);
```

Этот метод используется в основном с XML-документами (см. также главу 18).

В DOM Level 2 Views доступно новое свойство `defaultView`, содержащее указатель на окно (или фрейм), которому принадлежит документ. В спецификации Views не указано, когда могут быть доступны другие представления, так что это единственное добавленное свойство. Оно поддерживается во всех браузерах, кроме Internet Explorer 8 и более ранних версий, где доступно эквивалентное свойство `parentWindow` (оно доступно также в Opera). Определить окно, которому принадлежит документ, можно следующим образом:

```
var parentWindow = document.defaultView || document.parentWindow;
```

В DOM Level 2 Core к объекту `document.implementation` добавлены методы `createDocumentType()` и `createDocument()`. Первый из них создает узел `DocumentType`, принимая три аргумента: тип документа и свойства `publicId` и `systemId`. Например, следующий код создает тип документа HTML 4.01 Strict:

```
var doctype = document.implementation.createDocumentType("html",
                  "-//W3C//DTD HTML 4.01//EN",
                  "http://www.w3.org/TR/html4/strict.dtd");
```

Тип существующего документа изменить нельзя, поэтому метод `createDocumentType()` полезен только при создании документов, для чего можно использовать метод `createDocument()`. Он принимает три аргумента: URI пространства имен элемента документа, имя тега элемента документа и тип нового документа. Например, так можно создать пустой XML-документ:

```
var doc = document.implementation.createDocument("", "root", null);
```

Этот код создает документ с элементом документа `<root>` без пространства имен и типа документа. Создать XHTML-документ можно следующим образом:

```
var doctype = document.implementation.createDocumentType("html",
                  "-//W3C//DTD XHTML 1.0 Strict//EN",
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd");

var doc = document.implementation.createDocument(
    "http://www.w3.org/1999/xhtml", "html", doctype);
```

В этом примере создается XHTML-документ с соответствующими пространством имен и типом документа. Он содержит единственный элемент `<html>`, все остальное содержимое нужно добавлять.

Модуль DOM Level 2 HTML добавляет к объекту `document.implementation` метод `createHTMLDocument()`, который создает полный HTML-документ с элементами `<html>`, `<head>`, `<title>` и `<body>`. Он принимает заголовок нового документа (строковое содержимое элемента `<title>`) и возвращает HTML-документ:

Листинг CreateHTMLDocumentExample.htm


```
var htmldoc = document.implementation.createHTMLDocument("New Doc");
alert(htmldoc.title);              // "New Doc"
alert(typeof htmldoc.body);        // "object"
```

Объект, созданный с помощью метода `createHTMLDocument()`, имеет тип `HTMLDocument` и содержит все его свойства и методы, включая свойства `title` и `body`. Этот метод поддерживается в Internet Explorer 9+, Firefox 4+, Safari, Chrome и Opera.

Изменения типа Node

Единственное изменение типа `Node`, не связанное с пространствами имен, — это новый метод `isSupported()`. Как и метод `hasFeature()` объекта `document.implementation`,

представленный в DOM Level 1, он указывает возможности узла. Этот метод принимает два аргумента: имя компонента и его версию. Если компонент реализован и может использоваться узлом, метод `isSupported()` возвращает `true`, например:

```
if (document.body.isSupported("HTML", "2.0")){
    // какие-то действия с использованием модуля HTML из DOM Level 2
}
```

Этот метод не очень полезен, потому что ему присущ тот же недостаток, что и методу `hasFeature()`: реализующие его разработчики сами решают, возвращать ли `true` или `false` для конкретного компонента. Для определения того, доступна ли какая-то конкретная функциональная возможность, лучше использовать механизм распознавания возможностей.

Для сравнения узлов в DOM Level 3 представлены методы `isSameNode()` и `isEqualNode()`. Они принимают узел и возвращают `true`, если полученный и опорный узлы одинаковы или эквивалентны соответственно. Узлы одинаковы, если они ссылаются на один объект, и эквивалентны, если они имеют один тип, содержат равные свойства `nodeName`, `nodeValue` и т. д., причем их свойства `attributes` и `childNodes` эквивалентны (содержат одинаковые значения в аналогичных позициях), например:

```
var div1 = document.createElement("div");
div1.setAttribute("class", "box");

var div2 = document.createElement("div");
div2.setAttribute("class", "box");

alert(div1.isSameNode(div1));     // true
alert(div1.isEqualNode(div2));    // true
alert(div1.isSameNode(div2));     // false
```

Здесь создаются два элемента `<div>` с одними и теми же атрибутами. Эти элементы эквивалентны, но не одинаковы.

В DOM Level 3 добавлены также методы присоединения дополнительных данных к DOM-узлам. Метод `setUserData()`, который назначает данные узлу, принимает три аргумента: задаваемый ключ, фактические данные (которые могут иметь любой тип) и функцию-обработчик. Назначить данные узлу можно следующим образом:

```
document.body.setUserData("name", "Nicholas", function(){});
```

Получить эти данные можно с помощью метода `getUserData()`, передав в него тот же ключ:

```
var value = document.body.getUserData("name");
```

Функция-обработчик для метода `setUserData()` вызывается при клонировании, удалении, переименовании узла с данными или при его импорте в другой документ, позволяя указать, что нужно сделать с данными пользователя в каждом

из этих случаев. Она принимает пять аргументов: число, задающее тип операции (1 — клонирование; 2 — импорт; 3 — удаление; 4 — переименование), ключ данных, значение данных, исходный узел и целевой узел. Исходный узел равен `null`, если узел удаляется, а целевой узел равен `null` при всех операциях, кроме клонирования. Затем можно указать, как следует сохранить данные. Вот пример:

Листинг UserDataExample.htm


```
var div = document.createElement("div");
div.setUserData("name", "Nicholas",
                function(operation, key, value, src, dest){
    if (operation == 1){
        dest.setUserData(key, value, function(){}); }
});

var newDiv = div.cloneNode(true);
alert(newDiv.getUserData("name"));    // "Nicholas"
```

Созданному элементу `<div>` здесь назначаются некоторые данные, включая пользовательские. Затем этот элемент клонируется методом `cloneNode()`, при этом вызывается функция-обработчик и данные автоматически назначаются клону. В результате метод `getUserData()` возвращает для клона то же значение, которое было назначено оригиналу.

Изменения фреймов

Обычные и встроенные фреймы, представленные типами `HTMLFrameElement` и `HTMLIFrameElement` соответственно, имеют в DOM Level 2 HTML новое свойство `contentDocument`. Оно содержит указатель на объект `document` или содержимое фрейма. Ранее было невозможно получить объект `document` непосредственно через элемент — приходилось использовать коллекцию `frames`. Теперь это можно сделать с помощью нового свойства:

Листинг IFrameElementExample.htm


```
var iframe = document.getElementById("myIframe");
var iframeDoc = iframe.contentDocument;  // не работает в IE до версии 8
```

Свойство `contentDocument` является экземпляром типа `Document` и может быть использовано как любой другой HTML-документ. Оно поддерживается в Opera, Firefox, Safari и Chrome. Internet Explorer до версии 8 не поддерживает свойство `contentDocument` во фреймах, но предоставляет свойство `contentWindow`, которое возвращает для фрейма объект `window` со свойством `document`. Таким образом, для доступа к объекту `document` встроенного фрейма во всех браузерах можно использовать следующий код:

Листинг IFrameElementExample2.htm

```
var iframe = document.getElementById("myIframe");
var iframeDoc = iframe.contentDocument || iframe.contentWindow.document;
```

Свойство `contentWindow` доступно во всех браузерах.

Доступ к объекту document обычного или встроенного фрейма регламентируется междоменными ограничениями. Попытка доступа к объекту document фрейма со страницей, загруженной из другого домена или поддомена либо по другому протоколу, приведет к ошибке.

Стили

Чтобы определить в HTML-коде стили, можно включить в файл внешнюю таблицу стилей с помощью элемента `<link>`, добавить в файл встроенные стили, используя элемент `<style>`, или задать стили для отдельного элемента, указав атрибут `style`. DOM Level 2 Styles предоставляет API для всех этих механизмов. Узнать, поддерживает ли браузер DOM Level 2 CSS, можно следующим образом:

```
var supportsDOM2CSS = document.implementation.hasFeature("CSS", "2.0");
var supportsDOM2CSS2 = document.implementation.hasFeature("CSS2", "2.0");
```

Доступ к стилям элементов

У любого HTML-элемента, поддерживающего атрибут `style`, есть свойство `style`, доступное в JavaScript-коде. Объект `style` является экземпляром типа `CSSStyleDeclaration` и содержит все стили, заданные с помощью HTML-атрибута `style`, но не каскадно применяемые стили из внешних или встроенных таблиц стилей. Каждое CSS-свойство, заданное в атрибуте `style`, представлено свойством объекта `style`. Поскольку в именах CSS-свойств слова разделяются дефисом (например, `background-image`), для использования в JavaScript-коде их нужно преобразовать в верблюжью нотацию. В следующей таблице указаны некоторые часто используемые CSS-свойства и соответствующие им свойства объекта `style`:

CSS-свойство	JavaScript-свойство
background-image	style.backgroundImage
color	style.color
display	style.display
font-family	style.fontFamily

Имена CSS-свойств преобразуются в имена JavaScript-свойств напрямую, исключая свойство `float`. Поскольку в JavaScript это зарезервированное слово, его нельзя использовать как имя свойства. В спецификации DOM Level 2 Style сказано, что соответствующее свойство объекта `style` должно называться `cssFloat`. Оно поддерживается в Internet Explorer 9, Firefox, Safari, Opera и Chrome. В Internet Explorer 8 и более ранних версий вместо этого используется свойство `styleFloat`.