Клиентский JavaScript

Клиентский JavaScript предназначен для того, чтобы превращать статические HTMLдокументы в интерактивные веб-приложения. Работа с содержимым веб страниц – это главное предназначение JavaScript.

Объект Window, представляющий окно, имеет свойство document, которое ссылается на объект Document. Этот объект Document и является объектом изучения. Также, кроме объекта Document будут рассмотрены его свойства и методы - объекты, которые представляют содержимое документа. HTML документы могут содержать текст, изображения, гиперссылки, элементы форм и т. д.

JavaScript сценарии могут обращаться ко всем объектам, которые представляют элементы документа, и манипулировать ими. Прямой доступ к объектам, представляющим содержимое документа, дает широчайшие возможности, но одновременно означает определенные сложности.

Динамическое содержимое документа

Исследование объекта Document начнется с метода write(), который позволяет за писывать содержимое в тело документа. Этот метод относится к унаследованной части DOM, и начиная с самых ранних версий JavaScript метод document.write() можно было использовать двумя способами. Первый и самый простой способ – вывести HTML текст из сценария в тело документа, анализ которого производится в текущий момент.

Рассмотрим следующий фрагмент, где в статический HTML документ с помощью метода write() добавляется информация о текущей дате:

```
<script>
    var today = new Date();     document.write("<p>Документ открыт: " + today.toString( ));
</script>
```

Необходимо отметить, что вывод текста в формате HTML в текущий документ возможен только в процессе его синтаксического анализа. То есть вызывать метод document.write() из программного кода верхнего уровня в теге <script> можно только в том случае, если исполнение сценария является частью процесса анализа документа. Если поместить вызов document.write() в определение функции и затем вызвать эту функцию из обработчика события, результат окажется неожиданным – фактически этот вызов уничтожит текущий документ и все содержащиеся в нем сценарии!

Во вторых, объект Document поддерживает еще один метод – writeln(), который идентичен методу write() за исключением того, что после вывода последнего аргумента добавляет символ перевода строки. Это может оказаться удобным, например, при выводе отформатированного текста в теге <pre>.

Свойства объекта Document

Рассмотрев «старейшие» методы объекта Document, перейдем к его «старейшим» свойствам: bgColor Цвета фона документа. Это свойство соответствует атрибуту bgcolor тега <body>.

- cookie Специальное свойство, позволяющее JavaScriptпрограммам читать и писать cookie файлы.
- domain Свойство, которое позволяет доверяющим друг другу вебсерверам, принадлежащим одному домену, ослаблять связанные с политикой общего происхождения ограничения на взаимодействие между их вебстраницами
- lastModified Строка, содержащая дату последнего изменения документа.
- location Устаревший синоним свойства URL.
- referrer URLадрес документа, содержащего ссылку (если таковая существует), которая привела броузер к текущему документу.
- title Текст между тегами <title> и </title> данного документа.
- URL Строка, задающая URLадрес, с которого был загружен документ.

Некоторые из этих свойств предоставляют информацию о документе в целом. Следующий фрагмент можно поместить в конец каждого вашего документа, чтобы автоматически предоставлять пользователю дополнительные сведения о документе, которые позволят судить о том, насколько устарел этот документ:

```
<hr><font size="1">

Документ:
<i><script>document.write(document.title);</script></i><br>

URL:
<i><script>document.write(document.URL);</script></i><br>

Дата последнего обновления:
<i><script>document.write(document.lastModified);</script></
i> </font>
```

Ранняя упрощенная модель DOM: коллекции объектов документа

В списке свойств объекта Document, который приводился в предыдущем разделе, отсутствуют важные категории свойств – коллекции объектов документа. Эти свойства, представляющие собой массивы, являются сердцем ранней объектной модели документа. С их помощью обеспечивается доступ к некоторым специальным элементам документа:

- anchors[] Массив объектов Anchor, представляющих якорные элементы документа. Якорный элемент (anchor) – это именованная позиция в документе, которая создается с помощью тега <a> и в которой вместо атрибута href определяется атрибут name. Свойство name объекта Anchor хранит значение атрибута name.
- applets[] Массив объектов Applet, представляющих Java-апплеты в документе.
- forms[] Массив объектов Form, представляющих элементы <form> в документе. Каждый объект Form обладает собственным свойством коллекцией с именем elements[], в котором содержатся объекты, представляющий элементы формы.
- images[] Массив объектов Image, представляющих элементы <img> в документе. Свойство src объекта Image доступно для чтения/записи. Запись строки URLадреса в это свойство вынуждает броузер прочитать и отобразить новое изображение (в старых версиях броузеров размеры нового изображения должны были сов падать с размерами оригинала). Программирование свойства src объекта Image позволяет организовать листание изображений и простейшие виды анимации.
- links[] Массив объектов Link, представляющих гипертекстовые ссылки в документе. Свойство href объекта Link соответствует атрибуту href тега <a>: в нем хранится строка URLадреса ссылки.

Как следует из имен этих свойств, они являются коллекциями всех ссылок, изображений, форм и прочего, что имеется в документе. Элементы этих массивов располагаются в том же порядке, в котором они находятся в исходном документе. Например, элемент document.forms[0] ссылается на первый тег <form> в доку менте, а document.images[4] – на пятый тег <img>.

Объекты, содержащиеся в этих коллекциях ранней версии DOM, доступны для JavaScriptпрограмм, но вы должны понимать, что ни один из них не дает возможности изменить структуру документа. Вы можете проверять адреса ссылок и изменять их, читать или записывать значения элементов форм и даже менять местами изображения, но вы не сможете изменить текст документа.

Именование объектов документа

Проблема использования числовых индексов при работе с коллекциями объектов документа состоит в том, что незначительные изменения, которые влекут за собой переупорядочивание элементов, могут привести к нарушениям в работе сценариев, опирающихся на исходный порядок следования элементов. Более надежное решение заключается в том, чтобы присваивать имена важным элементам документа и затем обращаться к ним по этим именам.

В ранней версии DOM для этих целей можно было задействовать атрибут name форм, элементов форм, изображений, апплетов и ссылок. Если атрибут присутствует, его значение используется в качестве имени соответствующего объекта. Например, предположим, что HTMLдокумент содержит следующую форму:

```
<form name="f1"><input type="button" value="Нажми меня"></form>
```

Допустим, что тег <form> является первым таким тегом в документе, тогда из JavaScriptсценария к получившемуся объекту Form можно обратиться любым из трех способов:

```
document.forms[0]    // По номеру формы внутри документа
document.forms.f1    // По имени, как к свойству
document.forms["f1"] // По имени, как к элементу массива
```

Фактически установка атрибута name в тегах <form>, <img> и <applet> (но не в теге <a>) позволяет обращаться к соответствующим объектам Form, Image и Applet (но не к объектам Link и Anchor), как к именованным свойствам объекта Document. То есть к форме можно обратиться так:

```
document.f1
```

События

Интерактивные JavaScript программы основаны на модели программирования, управляемого событиями. При таком стиле программирования веб-броузер генерирует событие, когда с документом или некоторым его элементом чтото происходит. Например, веб-броузер генерирует событие, когда завершает загрузку документа, когда пользователь наводит указатель мыши на гиперссылку или щелкает на кнопке в форме.

Если JavaScript-приложение интересует определенный тип события для определенного элемента документа, оно может зарегистрировать обработчик

события (event handler) – JavaScript-функцию или фрагмент JavaScript-кода для этого типа события в интересующем вас элементе.

Потом, когда возникает это событие, броузер вызовет код обработчика. Все приложения с графическим интерфейсом пользователя разработаны подобным образом: они ждут, пока пользователь что-нибудь сделает (т. е. ждут, когда произойдут события), и затем реагируют на его действия.

Различные типы происшествий генерируют различные типы событий. Наводя мышь на гиперссылку и щелкая кнопкой мыши, пользователь вызывает события разных типов. Даже одно и то же происшествие может возбуждать различные типы событий в зависимости от контекста, например, когда пользователь щелкает на кнопке Submit, возникает событие, отличное от события, возникающего при щелчке на кнопке Reset в форме.

Имеется довольно много различных атрибутов обработчиков событий, которые можно использовать в исходной модели обработки событий. В процессе развития клиентского JavaScript-программирования развивалась и поддерживаемая им модель обработки событий.

В каждую новую версию броузера добавлялись новые атрибуты обработчиков событий. И наконец, спецификация HTML 4 закрепила стандартный набор атрибутов обработчиков событий для HTML-тегов. В таблице ниже указано, какие HTML-элементы поддерживают каждый из атрибутов обработчиков событий. К элементам, не поддерживающим практически универсальные атрибуты обработчиков событий мыши, относятся <applet>, <bdo>,
, <font>, <frame>, <frameset>, <head>, <html>, <iframe>, <isindex>, <meta> и <style>.

Обработчик	Условия вызова	Поддержка
onabort	Прерывание загрузки изображения	<img>
onblur	Элемент теряет фокус ввода	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Элемент <select> или другой элемент потерял фокус и его значение с момента получения фокуса изменилось	<input>, <select>, <tex tarea>
onclick	Была нажата и отпущена кнопка мыши; следует за событием mouseup. Возвращает false для отмены действия по умолчанию (т. е. перехода по ссылке, очистки формы, передачи данных)	Большинство элементов

ondblclick Двойной щелчок Большинство элементов

onerror Ошибка при загрузке изображения <img>

onfocus Элемент получил фокус ввода <button>, <input>, <label>, <select>, <textarea>, <body>

onkeydown Клавиша нажата. Для отмены возвращает false Элементы формы и <body>

onkeypress Клавиша нажата и отпущена. Для отмены возвращает false Элементы формы и <body> onkeyup Клавиша отпущена Элементы формы и <body>

onload Загрузка документа завершена <body>, <frameset>, <img>

onmousedown Нажата кнопка мыши Большинство элементов

onmousemove Перемещение указателя мыши Большинство элементов

onmouseout Указатель мыши выходит за границы эле мента Большинство элементов

onmouseover Указатель мыши находится на элементе Большинство элементов

onmouseup Отпущена кнопка мыши Большинство элементов

onreset Запрос на очистку полей формы. Для пре дотвращения очистки возвращает false <form>

onresize Изменение размеров окна <body>, <frameset>

onselect Выбор текста <input>, <textarea> onsubmit Запрос на передачу данных формы. Чтобы предотвратить передачу, возвращает false <form>

onunload Документ или набор фреймов выгружен <body>, <frameset>

Обработчики событий как атрибуты

Как мы видели в примерах из предыдущих глав, обработчики событий (в исходной модели обработки событий) задаются в виде строк JavaScript-кода, присваиваемых в качестве значений HTMLатрибутам. Например, чтобы выполнить JavaScriptкод при щелчке на кнопке, укажите этот код в качестве значения атрибута onclick тега <input> (или <button>):

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

Значение атрибута обработчика события – это произвольная строка JavaScript кода. Если обработчик состоит из нескольких JavaScript-инструкций, они должны отделяться друг от друга точками с запятой. Например:

```
<input type="button" value="Щелкни здесь"
onclick="if (window.numclicks) numclicks++; else
numclicks=1;              this.value='Щелчок # ' +
numclicks;">
```

Если обработчик события требует нескольких инструкций, то, как правило, проще определить его в теле функции и затем задать HTMLатрибут обработчика события для вызова этой функции. Например, проверить введенные пользоватеем в форму данные перед их отправкой можно при помощи атрибута onsubmit тега <form>.

Проверка формы обычно требует, как минимум нескольких строк кода, поэтому не надо помещать весь этот код в одно длинное значение атрибута, разумнее определить функцию проверки формы и просто задать атрибут onsubmit для вызова этой функции. Например, если для проверки определить функцию с именем validateForm(), то можно вызывать ее из обработчика события следующим образом:

```
<form action="processform.cgi" onsubmit="return
validateForm();">
```

Помните, что язык HTML нечувствителен к регистру, поэтому в атрибутах обработчиков событий допускаются буквы любого регистра. Одно из распространенных соглашений состоит в употреблении символов различных регистров, при этом префикс «on» записывается в нижнем регистре: onClick, onLoad, onMouseOut и т. д.

Обработчики как свойства

На кнопку в этой форме можно сослаться с помощью выражения document.f1.b1, значит, обработчик события можно установить с помощью следующей строки кода:

```
document.f1.b1.onclick=function() { alert('Спасибо!');
};
```

Кроме того, обработчик события может быть установлен так:

```
function plead() { document.f1.b1.value += ", пожалуйста!"; } document.f1.b1.onmouseover = plead;
```

Обратите особое внимание на последнюю строку: здесь после имени функции нет скобок. Чтобы определить обработчик события, мы присваиваем свойству-обработчику события саму функцию, а не результат ее вызова. На этом часто «спотыкаются» начинающие JavaScript-программисты.

Регистрация обработчиков

В моде ли событий Level 2 обработчик события регистрируется для определенного элемента вызовом метода addEventListener() этого объекта. Этот метод принимает три аргумента.

Первый – имя типа события, для которого регистрируется обработчик. Тип события должен быть строкой, содержащей имя HTMLатрибута обработчика в нижнем регистре без начальных букв «on». Другими словами, если в модели Level 0 используется HTMLатрибут onmousedown или свойство onmousedown, то в модели событий Level 2 необходимо использовать строку "mousedown".

Второй аргумент addEventListener() представляет собой функцию-обработчик (или слушатель), которая должна вызываться при возникновении событий указанного типа. Когда вызывается ваша функция, ей в качестве единственного аргумента передается объект Event. Этот объект содержит информацию о событии (например, какая кнопка мыши была нажата) и определяет методы, такие как stopPropagation().

Последний аргумент метода addEventListener() – логическое значение. Если оно равно true, указанный обработчик события перехватывает события в ходе их распространения на этапе перехвата. Если аргумент равен false, значит, это нормальный обработчик события, который вызывается, когда событие происходит непосредственно в данном элементе или в потомке элемента, а затем всплывает обратно к данному элементу.

Например, вот как при помощи функции addEventListener() можно зарегистри ровать обработчик события submit элемента <form>:

```
document.myform.addEventListener("submit", function(e) { return validate(e.target); }, false);
```

Можно перехватить все события mousedown, происходящие внутри элемента с определенным именем, вызвав функцию addEventListener() следующим образом:

```
var mydiv = document.getElementById("mydiv");
mydiv.addEventListener("mousedown", handleMouseDown,
true);
```

Практическое задание

После загрузки страницы вывести в консоль информацию о всех ссылках, якорях, изображениях на странице.

Используя различные способы добавить несколько обработчиков событий различных типов, которые выводят в консоль информацию о произошедшем событии.

Через подмену src у <img> реализуйте простейшую анимацию или перелистывание картинок.