

Курс «Клиентский JavaScript»

Содержание

3	<u>Глобальный объект Window</u>
18	<u>Объект Document, базовая модель событий</u>
27	<u>Отдельные HTML элементы и работа с ними</u>
41	<u>Объектная модель документа (DOM)</u>
55	<u>Модель событий в объектной модели документа</u>

Глобальный объект Window

Клиентский JavaScript

По сути, понятие «клиентский JavaScript» состоит из нескольких вещей:

- Объект Window в качестве глобального объекта JavaScript и глобального контекста исполнения сценариев
- Объект Document, иерархия элементов HTML-страниц и работа с элементами (DOM)
- Модель событий

Ключевую роль играет объект Window. Именно он является тем самым глобальным объектом Global, в контексте которого и функционируют все сценарии на странице.

Второй по значимости для нас – объект Document, в рамках которого определена иерархия элементов HTML-страницы (документа, который загружен в окно). С этим объектом связана объектная модель документа DOM (Document Object Model), реализация которой предоставляет различные свойства и методы для работы с содержанием загруженного в окно документа. Именно DOM вместе с моделью событий и сделала JavaScript мощным средством реализации функциональности сайтов.

Краткая история развития DOM, текущая версия

По сути, версии стандарта DOM Level 0 и DOM Level 1 просто закрепили существующее положение дел в индустрии браузеров. По этой причине можно считать, что положения этих стандартов выполняются всеми браузерами. Эти возможности и будут изучаться в 5-7 модулях.

Более обширные возможности реализованы в стандарте DOM Level 2, который и является последней версией. Этот стандарт будет изучаться в 8-10 модулях.

Глобальный объект Window, его свойства и методы

Понимание объектов невероятно важно для клиентского JavaScript. Многоуровневая структура объектов работает простым образом – свойство одного объекта ссылается на другой объект. В свою очередь, свойство второго объекта может представлять собой следующий объект.

Это следует всегда помнить, рассматривая свойства объектов клиентского JavaScript.

В отличие от многих объектов, свойства и методы объекта Window мы будем рассматривать по частям, формируя их по смыслу.

Доступ к объектам среды браузера

Несмотря на то, что объект Window очень важен для нас, без некоторых его свойств связь JavaScript с нашими страницами была бы невозможна. Среди всех перечисленных свойств особенно важно свойство, что ссылается на объект Document. Именно этот объект позволяет работать с документом, загруженным в окно.

Свойства объекта Window

Название	Что означает
document	Ссылка на объект Document, в рамках которого реализована иерархия загруженного в окно документа
location	Ссылка на объект Location, отвечающий за адрес открытого в окне документа
history	Ссылка на объект History, связанный с кнопками «Вперед» и «Назад» браузера
navigator	Ссылка на объект Navigator, предоставляющий информацию о браузере
screen	Ссылка на объект Screen, предоставляющий информацию об экране монитора
frames	Ссылка на HTML-коллекцию фреймов, расположенных в загруженном документе (элементы frame и iframe)
status	Ранее позволяло управлять содержанием статусной строки окна В настоящее время не рекомендовано к использованию
defaultStatus	Ранее позволяло управлять содержанием статусной строки окна в моменты, когда курсор находился вне объектов, предусматривающих сообщение в статусной строке В настоящее время не рекомендовано к использованию

Location

Этот объект позволяет получить данные об адресе документа, который загружен в текущее окно, и загрузить новый документ.

Рассмотрим свойства на примере адреса документа

`http://www.google.com:80/search?q=javascript#test`

Свойства объекта Location

Название	Что означает	Данные примера
protocol	Протокол	http:
host	Хост и порт	www.google.com:80
hostname	Хост (без порта)	www.google.com
port	Номер порта	80
pathname	Строка пути (относительно хоста)	/search
search	Часть адреса после символа "?", включая символ "?"	?q=javascript
hash	Часть URL, которая идет после символа решетки "#", включая символ "#"	#test
href	Весь URL	http://www.google.com:80/search?q=javascript#test

Методы объекта Location позволяют загрузить новый документ в окно или перезагрузить текущий.

Методы объекта Location

Название	Аргументы	Что делает
assign(url)	url – адрес нового документа	Загружает в окно новый документ В истории посещений окна появляется новый переход
reload(b)	b – логическое значение	Перезагружает документ по текущему URL Если параметр r равен true, то документ забирается с сервера Если параметр не задан или равен false, документ может быть полностью или частично взят из кеша браузера
replace(url)	url – адрес нового документа	Загружает в окно новый документ В отличие от метода assign(), в истории посещений окна новый переход не добавляется Перейти по кнопке «Назад» на предыдущую страницу не получится
toString()	–	Возвращает строковое представление объекта Location Результат аналогичен значению location.href

На практике вместо метода `assign(url)` применяют более простой подход – присваивают свойству `location` требуемый URL нового документа. Несмотря на некорректность присваивания объекту строкового значения, этот способ укоренился в практике.

Это связано, в том числе, и с тем, что политика безопасности JavaScript запрещает доступ к свойствам и методам глобального объекта другого окна, если он загружен с другого домена. Это ограничение и понятие «происхождение документа» мы рассмотрим ниже.

Предположим, на странице есть `iframe`, в который загружен документ с другого домена, и мы имеем ссылку на это внутреннее окно в качестве значения переменной `i`.

```
i.location.assign('новый_адрес'); //выдаст "permission denied"  
i.location = 'новый_адрес'; //операция пройдет успешно
```

History

Этот объект позволяет получить общую информацию об истории посещений в рамках окна. Эта информация малоинформативна и просто бесполезна. На заре появления этого объекта он представлял собой массив ссылок посещенных страниц, однако очень скоро такая информация из объекта исчезла в силу конфиденциальности личной информации пользователя.

Свойства объекта History

Название	Что означает
length	Количество страниц, посещенных в данном окне

Методы объекта History

Название	Аргументы	Что делает
back()	–	Загружает предыдущий документ Переход в истории посещений на 1 шаг назад Если переходить некуда, ничего не происходит
forward()	–	Загружает следующий документ Переход в истории посещений на 1 шаг вперед Если переходить некуда, ничего не происходит
go(n)	n – количество шагов (целое число)	Загружает предыдущий или последующий документ, отстоящий в истории от текущего документа на заданное количество шагов go(-1) равносильно методу back() go(1) равносильно методу forward() Если переходить некуда, ничего не происходит

Продолжая разговор о полезности этого объекта, отметим, что применение методов этого объекта оправдано только тогда, когда заранее известно, по какому пути пользователь попадает на эту страницу. В противном случае использование методов приведет совсем не к тому результату.

Например, разработчик планировал, что на страницу товара попадают со страницы рубрики каталога, и поставил ссылку «Вернуться в каталог», которая использует back(). На практике посетитель попал на страницу товара со страницы результатов поиска, скажем, Yandex и, в лучшем случае, вернется на страницу результатов поиска.

По этой причине использование этих методов может быть оправдано на страницах, чье местонахождение однозначно определено, и на которые невозможно попасть, минуя некоторые стадии.

Navigator

Объект предоставляет информацию о браузере, в котором открыто окно. Не так давно этот объект использовался активно, однако, в настоящее время его значение для программистов очень и очень небольшое.

Свойства объекта Navigator

Название	Что означает
appName	Кодовое имя браузера Практически все браузеры вернут "Mozilla"
appVersion	Имя браузера
userAgent	Внутренний номер версии браузера и другая служебная информация Внутренний номер, как правило, содержит не тот номер, который принят для обозначения и виден пользователю
cookieEnabled	Информация, содержащаяся в HTTP заголовке USER-AGENT Представляет собой комбинацию appName (или appVersion) и appVersion
platform	Если браузер поддерживает запись cookie, то вернет true В обратном случае false
onLine	Аппаратная платформа, на которой работает браузер
plugins	Если включена опция «работать автономно» (меню File браузера), вернет false В противном случае вернет true
mimeTypes	Массив подключенных плагинов В IE не поддерживается...
	Массив поддерживаемых mime-types В IE не поддерживается...

Методы объекта Navigator

Название	Аргументы	Что делает
javaEnabled()	–	Возвращает true, если в окне могут быть выполнены java-апплеты В противном случае вернет false

Написание кроссбраузерного кода

В эпоху, когда реализация JavaScript сильно отличалась в различных браузерах, программисты ориентировались на определение имен и версий браузеров. В настоящее время используются другие подходы. Подробнее об этом будет говориться на следующих занятиях, но вкратце можно сейчас описать методы написания кроссбраузерного кода.

Как правило, для решения той или иной задачи не нужно проверять имя и версию браузера (кроме исключительных случаев). Нужно ориентироваться на поддержку стандартизированных свойств или методов и проверять именно ее. В противном случае используем те методы и свойства, что являются уникальными для тех или других браузеров.

Как правило, для таких решений используются инструкции if/else if/else или try/catch.

Screen

С помощью этого объекта можно получить информацию о разрешении монитора и размерах рабочей области (например, размер экрана за вычетом панелей задач в Windows).

Свойства объекта Screen

Название	Что означает
width	Ширина экрана монитора в пикселях
height	Высота экрана монитора в пикселях
availWidth	Ширина рабочей области экрана монитора в пикселях
availHeight	Высота рабочей области экрана монитора в пикселях

Свойства этого объекта могут быть полезны при позиционировании открываемого окна.

status и defaultStatus

Эти свойства позволяли раньше управлять содержанием статусной строки (текст в полоске внизу браузера).

Их отличие состоит в том, что status изменяет текст на все время, а defaultStatus – только в моменты, когда курсор находится вне объектов, требующих изменения статусной строки (например, гиперссылок).

```
status = 'новый текст'; //слева в строке статуса появлялся заданный текст
```

В настоящее время только в IE можно управлять текстом в статусной строке. Причина в том, что это свойство позволяло маскировать истинный адрес гиперссылки и вводить в заблуждение посетителей.

В настоящее время использовать это свойство не нужно.

Работа с окнами

Frames и понятие HTML коллекции

Свойство frames представляет собой ссылку на особый объект – html-коллекцию окон, содержащихся в главном окне.

Окно внутри другого окна может оказаться благодаря использованию 2 элементов:

- frameset, внутри которого содержатся элементы frame
- iframe внутри обычной страницы

Что же такое html-коллекция? С одной стороны она очень похожа на массив – у нее есть отдельные элементы, доступные по индексу и свойство length. С другой стороны, у коллекции отсутствуют методы массивов.

Кроме того, доступ к отдельным элементам в любой html-коллекции может осуществляться как к свойствам объекта – по имени. В качестве названия свойства выступает атрибут name у элементов. В случае свойства frames – значения атрибута name у элементов frame/iframe.

Следует понимать, что получая ссылки на наборы html-элементов, мы имеем дело с html-коллекциями. Мы в дальнейшем будем иногда называть такие наборы массивами, но подразумеваем, что это – коллекции.

Обращение к внутреннему окну через свойство frames

Предположим, что внутри обычной страницы есть 2 элемента iframe, в каждый из которых загружен свой документ. В html-коде iframe с name="top" стоит выше.

```
<iframe name="top" src="top.html"></iframe>
<iframe name="bottom" src="bottom.html"></iframe>
```

Обратиться к окну, сформированному iframe с name="top" можно 3-мя способами:

```
frames[0] //как к элементу массива
frames['top'] //как к свойству объекта
frames.top //как к свойству объекта
```

Отношения между окнами

В работе с фреймовыми структурами и новыми открытыми окнами, нам потребуется взаимодействие между этими окнами. Для примера можно привести несколько задач:

- Из открытого с помощью window.open() окна запустить глобальную функцию окна, методом которого было открыто новое окно
- Из одного фрейма обращаться к глобальным переменным другого фрейма
- Во вложенной функции получить значение глобальной переменной, которая имеет то же имя, что и локальная

Для решения этих задач существуют соответствующие свойства.

Свойства объекта Window – отношения окон между собой

Название	Что означает
window	Ссылка на само окно
self	То же самое, что и window
parent	Ссылка на родительское окно Для окна, сформированному с помощью iframe/frame, – окно, в которое загружен документ с iframe
top	Ссылка на окно самого верхнего уровня Если есть цепочка окон, доступных по parent, top представляет собой последнее окно в этой цепочке Для единственного вложенного окна его свойства parent и top равны
opener	Окно, открытое как самостоятельное с помощью метода window.open(), свойство opener ссылается на окно, открывшее его
name	Атрибут у html-элемента, формирующего окно frame/iframe Также может быть установлен как свойство объекта Window

Теперь можно с уверенностью сказать, что глобальные переменные представляют собой свойства глобального объекта Window, а функции, декларированные в глобальном контексте, являются методами этого глобального объекта.

```
var x = 5;
function y() {
    var x = 3;
    alert(x); //выведет 3 – значение локальной переменной
    alert(window.x); //выведет 5 – значение глобальной переменной
}
y();
```

По сути, найти разницу между parent и top можно только в сложных фреймовых структурах.

Например, в iframe загружена страница, содержащая еще один элемент iframe, в который загружена третья страница. В этом случае свойство parent окна третьей страницы будет ссылаться на окно второй, а свойство top третьей страницы будет ссылаться на первое окно.

По цепочке parent можно добраться до окна самого верхнего уровня. То есть в рассматриваемом случае для объекта Window третьего окна:

```
parent.parent == top
```

Диалоговые окна

У объекта Window есть 3 вида диалоговых окон, являющихся модальными. Модальными эти окна называются по причине перехвата управления – невозможно воспользоваться ни одним элементом управления браузера, пока не будет закрыто такое окно. Кроме того, исполнение программного кода «замораживается», пока модальное окно не будет закрыто.

Эти диалоговые окна предназначены для вывода информации, ввода ее и выбора пользователем одного из вариантов. Каждое из этих окон формируется с помощью вызова одного из методов объекта Window.

Методы объекта Window – диалоговые окна

Название	Аргументы	Что делает
alert(s)	s – строка для вывода	Выводит переданную строку Если передать объект (переменную), он будет трансформирован в строку
confirm(s)	s – строка (вопрос)	Выбор ответа на переданный вопрос – кнопки «Ок», «Нет» При нажатии на кнопку «Ок» возвращает true При нажатии на кнопку «Отмена» возвращает false
prompt(s1, s2)	s1 – строка (вопрос) s2 – строка (первоначальный вариант ответа)	Предполагает ввод пользователем ответ на вопрос Возвращает введенную строку при нажатии на кнопку «Ок» При нажатии на кнопку «Отмена» вернет null

Практика хорошего программирования предполагает использование таких «жестких» методов скорее, в качестве отладки кода, чем для использования в реальных проектах.

Таймеры

До этого момента весь написанный нами код исполнялся непосредственно при загрузке документа. Пара методов глобального объекта позволяет запустить исполняемый код с задержкой, определяемой в них.

Методы объекта Window – таймеры

Название	Аргументы	Что делает
<code>setTimeout(c, t)</code>	<code>c</code> – исполняемый код <code>t</code> – время задержки в миллисекундах	Выполняет исполняемый код один раз с указанной задержкой и возвращает ссылку на отложенное выполнение
<code>clearTimeout(x)</code>	<code>x</code> – ссылка на отложенный вызов	Отменяет отложенное выполнение кода
<code>setInterval(c, t)</code>	<code>c</code> – исполняемый код <code>t</code> – время задержки в миллисекундах	Выполняет исполняемый код постоянно с указанной задержкой и возвращает ссылку на отложенное выполнение
<code>clearInterval(x)</code>	<code>x</code> – ссылка на отложенный вызов	Отменяет отложенное выполнение кода

Исполняемый код и `this`

Прежде чем разбираться с работой методов, определим сначала, что же представляет собой исполняемый код. Это может быть непосредственно определение функции:

```
setTimeout(function(){исполняемый_код}, 1000);
```

Это может быть строка. В этом случае по отношению к строке будет неявно выполнен метод `eval()`:

```
setTimeout('исполняемый_код', 1000);
```

Можно в качестве строки написать вызов функции

```
function test(){исполняемый_код}  
setTimeout('test()', 1000);
```

В качестве аргумента, как мы делали и раньше, можно указать имя функции или переменной, чье значение представляет строку. В этом случае, как мы уже знаем, программа подставит значения переменных.

```
function test(){исполняемый_код}  
setTimeout(test, 1000);
```

```
var test = 'исполняемый_код';  
setTimeout(test, 1000);
```

Важно понимать, что исполняться код будет всегда в контексте глобального объекта, и ключевое слово `this` будет ссылаться на глобальный объект.

Отмена вызова

Для обоих методов есть пара методов, позволяющих отменить отложенный вызов исполнения кода. Для этого при создании отложенного вызова следует присвоить переменной результат его выполнения. В этом случае в переменной будет храниться ссылка на этот отложенный вызов и с помощью обратного метода можно его отменить.

```
var x = setTimeout('исполняемый_код', 1000); //создаем отложенный вызов
clearTimeout(x); //вызов не будет выполнен
```

Применение методов

Метод `setTimeout()` логично вызывать, когда исполняемый код сам определяет условие повторного вызова этого кода. Например, счетчик для заранее определенного количества вызовов.

```
var count = 0;
function test(){
  if (count < 10) {
    setTimeout(arguments.callee, 1000);
  }
  count++;
}
setTimeout(test, 1000);
```

Метод `setInterval()` лучше использовать, когда прерывание запусков кода определяется внешними условиями.

Интерес представляет отложенный вызов с 0-й задержкой. В этом случае вызов сработает не сразу, а тогда, когда все остальные процессы, необходимые к завершению, будут выполнены.

Открытие новых окон

Для загрузки новых служебных страниц или показа фотографий раньше использовался метод, позволяющий открыть новое независимое окно с нужными размерами и положением. И, хотя в настоящее время для решения большинства задач есть менее brutальные методы, мы постоянно встречаем открытие новых окон для показа рекламы.

Методы объекта Window – открытие окон

Название	Аргументы	Что делает
open(url, name, p, l)	url – адрес документа нового окна (строка, необязательный аргумент) name – имя нового окна (строка, необязательный аргумент) p – параметры окна (строка, необязательный аргумент) l – добавлять в историю новую запись или нет (логическое значение, необязательный аргумент)	Открывает новое окно и возвращает на него ссылку
close()	–	Закрывает открытое окно
focus()	–	Передает фокус окну и делает его автоматически видимым
blur()	–	Снимает фокус с окна

Параметры открываемого окна перечисляются через запятую в виде пар «имя=значение»

Параметры открываемого окна

Название	Возможное значение	Что означает
width	цифра	Ширина внутренней части окна
height	цифра	Высота внутренней части окна
left	цифра	Координата левого верхнего угла окна по оси X
top	цифра	Координата левого верхнего угла окна по оси Y
location	0 (no) 1 (yes)	Отображение строки с адресом
menubar	0 (no) 1 (yes)	Панель меню
scrollbars	0 (no) 1 (yes)	Полосы прокрутки
toolbar	0 (no) 1 (yes)	Панель инструментов браузера
status	0 (no) 1 (yes)	Статусная строка
resizable	0 (no) 1 (yes)	Возможность изменения размеров окна

Для параметров, значение которых мы хотим установить yes или 1 (что одно и то же), можно написать просто название параметра.

Как применять

Применяя метод `window.open()`, разумно присвоить переменной результат его выполнения – ссылку на открытое окно. В этом случае мы получаем возможность оперировать открытым окном, изменять его, закрывать, и просто получить доступ к его глобальным переменным и функциям (его свойствам и методам).

Метод не передает автоматически фокус на новое окно, поэтому после его открытия следует передать ему фокус явно.

```
var x = window.open('', '', 'width=300, height=300, location=0');  
x.focus(); //передали фокус на новое окно
```

Когда нам нужно закрыть это окно, мы применяем соответствующий метод

```
x.close();
```

Этот метод при попытке закрыть основное окно вызовет ошибку или предупреждение. По этой причине метод `close()` применяется только для окон, открытых с помощью `window.open()`.

С помощью свойства `closed` можно всегда проверить – закрыто ли пользователем или нами новое окно.

Свойства объекта Window – открытие окон

Название	Что означает
<code>closed</code>	Возвращает <code>true</code> , если окно закрыто В противном случае вернет <code>false</code>

После открытия нового окна мы можем управлять им – передвигать, изменять размеры и прокручивать содержимое, вызывая методы этого нового окна.

Методы объекта Window – управление открытыми окнами

Название	Аргументы	Что делает
<code>moveTo(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Передвигает окно в заданную точку относительно левого верхнего угла экрана
<code>moveBy(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Передвигает окно на заданное расстояние относительно текущего положения окна
<code>resizeTo(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Изменяет размеры окна на заданные
<code>resizeBy(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Изменяет размеры окна на заданную величину относительно текущих размеров
<code>scrollTo(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Пытается прокрутить содержимое окна до заданной точки (совмещает точку и левый верхний левый угол окна)
<code>scrollBy(x, y)</code>	<code>x</code> – координата по оси X <code>y</code> – координата по оси Y	Пытается прокрутить содержимое окна на заданное расстояние относительно текущего положения

Не стоит пытаться изменять размеры и положение основного окна.

Во-первых, в браузере может быть открыто много вкладок и другие документы явно не нуждаются в таком поведении браузера.

Во-вторых, с помощью настроек браузера пользователь может запретить изменение текущего окна.

Ну и, в-третьих, такое поведение смотрится неприятно, что может просто оттолкнуть посетителей от сайта.

Особенности

Большинство методов и свойств объекта Window записываются без указания window, например, не window.screen, а просто screen. Методы open() и close() – отдельная история. Методы с такими же названиями есть у объекта document, поэтому их мы записываем полностью.

Некоторые общие методы объекта Window

Фактически, кроссбраузерно поддерживается только один метод.

Общие методы объекта Window

Название	Аргументы	Что делает
print()	–	То же самое, что кнопка «Отправить на печать»

Объект Document, базовая модель событий

Важнейшим из свойств объекта Window является document – ссылка на объект Document. Именно этот документ является основой для объектной модели документа DOM. В DOM версий 0 и 1 функциональность работы с документом реализована слабо, однако мы рассмотрим вначале ее, затем перейдя к DOM версии 2.

Объект Document

Часть устаревших свойств мы не будем рассматривать (bgColor, location, URL). Они не используются на практике.

Свойства объекта Document

Название	Что означает
lastModified	Строка даты последнего изменения файла
referrer	URL-адрес страницы, с которой пришел пользователь по ссылке
title	Текст элемента <title> на странице
domain	Свойство, позволяющее ослабить требования политики общего происхождения
cookie	Свойство, позволяющее писать и читать информацию в специальных cookie-файлах

Свойство referrer активно используется в скриптах по сбору и анализу статистики (например, google.analytics).

Если с первыми тремя свойствами все достаточно просто, то последние два представляют собой более обширные области.

domain, «происхождение» и политика безопасности

У любого документа, загруженного в окно, так называемое «происхождение» определяется протоколом и адресом хоста (домен, поддомен, порт).

Во взаимоотношениях окон как для фреймовой структуры, так и для открытых с помощью window.open(), ключевую роль играет домен. Для документов, загруженных с разных доменов, доступ к свойствам другого окна сильно ограничен. Фактически, мы можем только загрузить новый документ в окно, в который загружен документ с другого домена. Доступ к такому документу просто невозможен.

Это и есть политика безопасности в части происхождения документов.

Однако, документы с одного домена могут иметь домены 3-го и выше уровней (поддомены), а также порт (что встречается значительно реже). Политика безопасности для такой ситуации запрещает доступ к свойствам одного документа из окна, в котором открыт другой. Но в этом случае мы можем снизить ее уровень, воспользовавшись свойством domain.

Предположим, мы имеем страницу, на которой расположен iframe. В главное окно загружен документ с происхождением http://some_domain.ru. В iframe загружен документ с происхождением http://dir.some_domain.ru.

В обоих окнах нужно установить:

```
document.domain = 'some_domain.ru';
```

Этими действиями мы снизили требования политики безопасности и теперь окна могут обращаться к свойствам документов, загруженных в них, а также к свойствам и методам окон.

Для более сложных фреймовых структур такое действие следует произвести для каждого окна.

cookie

Это свойство позволяет запомнить, а впоследствии прочитать некоторую информацию. Это используется в реализации схемы распознавания посетителя (например, в системах с авторизацией).

Это – единственное свойство, которое позволяет записывать файлы на жесткий диск. Кроме JavaScript, использовать это свойство могут и серверные скрипты.

document.cookie представляет собой строку текста, записанную определенным образом – пары «свойство=значение» разделены точкой с запятой “;” (comma separated). При этом у cookie есть 4 свойства, влияющих на время жизни, видимость и безопасность.

Свойства cookie

Название	Возможное значение	Что означает
expires	строковое представление даты	Дата, когда истекает время жизни файла cookie В настоящее время предпочтительнее использовать max-age
max-age	цифра – количество секунд	Время жизни файла cookie в секундах Становится все более распространенным
path	строка – путь от корня сервера	Определяет видимость cookie для страниц, находящихся выше по иерархии
domain	адрес домена	Определяет видимость cookie для страниц, находящихся в других поддоменах
secure	true false	Защищенность при передаче Если нет или false – можно передавать данные cookie по протоколу http Если true, то данные передаются только по протоколу https

Запись

Эти свойства записываются ровно так же, как и пользовательские. Свойство path чаще всего устанавливают в значение "/".

```
document.cookie = 'path=/' ;  
document.cookie = 'domain=some_domain.ru' ;  
document.cookie = 'my_product_version=5' ;
```

Имена свойствам дают, как правило, так же, как и переменным. А для записи значений свойства, как правило, их кодируют в шестнадцатеричные последовательности с помощью метода глобального объекта `encodeURIComponent()`. Это связано с тем, что в значениях могут встретиться знаки ";", "=" . Кроме этой причины существовали проблемы в некоторых браузерах при чтении значений, в которых встречались не латинские буквы.

Чтение

Для получения данных из cookie-файла следует работать со значением `document.cookie`, как со строкой. Для нахождения информации и ее выделения используют, например, методы строки `indexOf()`, `split()`.

Однако, самый быстрый результат можно получить, используя регулярные выражения, их метод `exec()` или метод строк `match()`.

После получения значения его декодируют с помощью метода глобального объекта `decodeURIComponent()`.

Ограничения

Следует помнить, что браузеры не обязаны хранить файлы cookie:

- всего более 300
- более 20 на одно доменное имя
- более 4 кб на один файл

Запись содержания документа

Для записи содержания документа используются специальные методы объекта Document.

Методы объекта Document – запись содержания документа

Название	Аргументы	Что делает
open()	–	Открывает поток документа на запись
write(s1, ...)	s1, ... – одна или несколько строк, перечисленных через запятую	Записывает переданный HTML-код в документ
close()	–	Закрывает поток документа

Метод `document.write()` – то самое, ради чего браузер ждет загрузки скриптов перед тем, как показать страницу. Этот метод может дописать контент.

Следует понимать, что применение этого метода после загрузки страницы переписывает документ начисто и полностью.

Если в момент загрузки страницы запись потока документа открыта, и закрывается самим браузером по завершению формирования страницы, то вызов методов `document.open()` и `document.close()` не нужен.

После загрузки страницы метод `document.open()` вызовется автоматически, когда запустится `document.write()`, а вот `document.close()` – нет. Поэтому лучше привыкнуть к нормальному программированию и вызывать оба метода при записи содержания после загрузки страницы.

Доступ к отдельным элементам страницы

Некоторые элементы страницы доступны в качестве HTML-коллекций, на которые ссылаются соответствующие свойства объекта Document.

Свойства объекта Document

Название	Что означает
forms	Коллекция форм <code><form></code>
images	Коллекция изображений <code></code>
links	Коллекция гиперссылок <code></code>
anchors	Коллекция якорей <code></code>
applets	Коллекция java-апплетов <code><applet></code>
embeds	Коллекция элементов <code><embed></code>

В настоящее время после реализации DOM Level 2 на практике активно используют только коллекцию `forms`.

Некоторые из этих свойств морально устарели, как например, `anchors`. Использование элемента `<a>` для якорей признано не рекомендованным.

Как это было рассмотрено для коллекции `frames`, для форм и остальных коллекций действует правило доступа к отдельному элементу – по индексу (порядок расположения в коде страницы) или с помощью атрибута `name`.

Предположим, есть форма поиска, стоящая в коде первой и имеющая атрибут `name` со значением `search`.

```
document.forms[0] //как к элементу массива
document. forms['search'] //как к свойству объекта
document. forms.search //как к свойству объекта
```

Базовая или исходная модель событий

Без модели событий реализовать хоть какую-то функциональность было бы невозможно. Код, написанный в глобальном контексте, выполняется при загрузке страницы. Введение событий в HTML-страницы позволил сделать страницы интерактивными, что и привлекает внимание к JavaScript.

Событие – отклик страницы на некоторое действие пользователя или изменение состояния страницы/окна. Для того чтобы событие вызвало реакцию в коде JavaScript, требуется поставить этот код, называемый обработчиком, в соответствие с этим событием. Установление этой связи называется регистрацией обработчика события.

В первых версиях модели событий была закреплена регистрация обработчика события в качестве атрибута HTML-элемента.

```
<body onload="alert('Hooray!');">
```

Все события имеют разные места их возникновения. Это означает, что не во всех элементах могут возникнуть любые события.

Список событий и элементы, в которых они могут возникнуть

Название	Элемент	Что означает
onclick	Большинство элементов	Была нажата и отпущена кнопка мыши на одном и том же элементе Для отмены возвращает false
ondblclick *	Большинство элементов	Двойной щелчок
onmousedown	Большинство элементов	Была нажата кнопка мыши
onmouseup	Большинство элементов	Была отпущена кнопка мыши
onmousemove	Большинство элементов	Перемещение указателя мыши
onmouseover	Большинство элементов	Указатель мыши попадает на элемент
onmouseout	Большинство элементов	Указатель мыши выходит за границы элемента
onkeydown	<body>, <input>, <textarea>	Клавиша нажата Для отмены возвращает false
onkeypress	<body>, <input>, <textarea>	Клавиша нажата и отпущена Для отмены возвращает false
onkeyup	<body>, <input>, <textarea>	Клавиша отпущена
onfocus	<body>, <a>, <area>, <button>, <input>, <label>, <select>, <textarea>	Элемент получил фокус ввода
onblur	<body>, <a>, <area>, <button>, <input>, <label>, <select>, <textarea>	Элемент теряет фокус ввода
onload	<body>, <frameset>, <iframe>, , <object>	Загрузка документа завершена
onunload	<body>, <frameset>	Документ или набор фреймов выгружен

onabort	, <object>	Прерывание загрузки изображения
onerror	<body>, <frameset>, , <object>	Ошибка при загрузке изображения
onresize	<body>, <frameset>, <iframe>	Изменение размеров окна
onreset	<form>	Запрос на очистку полей формы Для предотвращения очистки возвращает false
onsubmit	<form>	Запрос на передачу данных формы Чтобы предотвратить передачу, возвращает false
onselect	<input>, <textarea>	Выбор текста
onchange	<input>, <select>, <textarea>	Элемент потерял фокус, и его значение с момента получения фокуса изменилось
onscroll	<body>, элементы с прокруткой	Внутреннее содержание элемента было прокручено с помощью линейки или колеса мышки

* Поддержка события ondblclick исключена в стандарте DOM 2.

Сиреневым цветом помечены события, для которых можно отменить действия по умолчанию

Отмена действия по умолчанию

Нужно обратить внимание, что некоторые события имеют действия по умолчанию – click для гиперссылок и кнопок формы, очистка и отправка формы, нажатие клавиш. Для того чтобы отменить событие, нужно при регистрации использовать инструкцию return.

В качестве обработчика можно писать код прямо в атрибуте, обозначающем событие. Поскольку значением атрибута может быть только строка, инструкции разделяются точкой с запятой. Комментировать отдельные инструкции можно только с помощью /**/.

```
<form onsubmit="alert('Не отправляется');return false;">
```

Такая практика и в ранние времена признавалась плохой. Поэтому применялся вызов функций, определенных в глобальном контексте.

```
function test() {
    alert('Не отправляется');
    return false;
}
...
<form onsubmit="return test();">
```

Событие ожидает, что функция test() вернет ей значение, которое может быть интерпретировано как логическое. А функция test(), в свою очередь, заканчивает работу инструкцией return, которая возвращает ожидаемое значение.

Передача ссылки на элемент, в котором возникло событие

Внутри значения атрибута, обозначающего событие, ссылка на этот элемент доступна по ключевому слову `this`.

Если в глобальной функции нам потребуется ссылка на элемент, в котором произошло событие, необходимо передать в эту функцию ссылку на элемент с помощью ключевого слова `this`.

```
function test(e) {  
    alert(e.name); //выведет "search"  
    return false;  
}  
...  
<form name="search" onsubmit="return test(this);">
```

Регистрация обработчиков в качестве свойства элемента

Все атрибуты HTML-элементов доступны в качестве свойств этих объектов (ссылок на эти элементы). Это означает, что для первого элемента `` в HTML-коде

```
<img width="200" height="200">  
document.images[0].width = '300'; //теперь ширина картинки 300 пикселей
```

Однако, чем отличается регистрация обработчика от другого атрибута? Ничем. Поэтому, можно поступить ровно так же, как и с картинкой.

```
<form name="search">  
...  
document.forms.search.onsubmit = test;  
function test() {  
    alert(this.name); //выведет "search"  
}
```

Стоит обратить внимание, что ссылку на элемент передавать не нужно – она доступна по ключевому слову `this`. Почему? Потому, что мы вызываем функцию `test()` в качестве метода объекта, в данном случае ссылки на HTML-элемент.

Несколько моментов, на которые следует обратить внимание.

Обращение к любому HTML-элементу должно происходить тогда, когда этот элемент уже обработан анализатором HTML-кода. Это значит, что производить действия можно или в скрипте, который стоит в коде после этого элемента, либо из любого места после наступления события загрузки страницы `onload`.

В качестве обработчика события нужно ставить не вызов функции `test()`, а ее имя. В этом случае код функции будет выполнен по наступлению события. Впрочем, такое присваивание нам уже не впервой.

Регистрация обработчиков событий в коде скрипта, передача ссылки на событие

Вызов исполняемого кода с помощью javascript:

Помимо включения кода скрипта с помощью элемента `<script>`, установки исполняемого кода в качестве значения атрибута события в HTML-элементах, есть возможность написания исполняемого кода после псевдопротокола "javascript:". Этот псевдопротокол можно использовать там, где может стоять URL-адрес.

Например, просто вбить в адресную строку браузера:

```
javascript:alert('Hello World!');
```

Этот псевдопротокол может применяться в значениях атрибута href у гиперссылок и target у форм, адреса нового окна при использовании метода window.open().

Инструкции пишутся в одну строку и разделяются точкой с запятой.

Если последняя инструкция может быть преобразована в строку, она используется для записи нового содержания документа. Старое содержание затирается, как при использовании метода document.write().

```
javascript:var now = new Date(); "<h1>Время:</h1>" + now;
```

Для предотвращения такого поведения последней инструкцией записывают "void(0)".

```
javascript:var now = new Date(); "<h1>Время:</h1>" + now; void 0;
```

В настоящее время этот протокол использовать не рекомендуется.

Отдельные HTML элементы и работа с ними

В рамках первых версий DOM для отдельных HTML-элементов были реализованы особые свойства и методы. В число этих элементов входят:

- Формы и элементы форм
- Таблицы
- Изображения

Формы

Коллекция `document.forms`

Формы доступны как элементы HTML-коллекции `document.forms`.

Как и в рассмотренной выше коллекции `frames` к формам можно обратиться по их индексу, который зависит от расположения формы в коде HTML-страницы. Как правило, формы содержат атрибут `name`, который позволяет обратиться к форме, как к свойству этой коллекции.

```
<form name="search">
...
var f = document.forms.search;
```

Коллекция `forms`, равно как и другие коллекции документа, обладает особенностью – все атрибуты `name` у форм доступны как свойства объекта `document`. Рассмотренный выше пример можно переписать следующим образом:

```
<form name="search">
...
var f = document.search;
```

На практике используют чаще именно такой способ доступа к форме. Следует учесть этот момент в том плане, что нужно стараться не использовать в качестве значения атрибута `name` формы названий свойств и методов объекта `document`.

Атрибуты HTML элементов

На примере форм мы впервые сталкиваемся с тем, что атрибуты HTML элементов доступны нам как свойства объекта – ссылки на этот элемент. Это означает, что в любой момент времени после получения доступа к HTML элементу можно изменить его атрибуты.

Например, у форм есть атрибут `method`. Его можно переписать следующим образом:

```
<form name="search" method="get">
var f = document.search;
f.method = 'post';
```

Это работает практически для всех HTML элементов страниц, к которым у нас будет доступ. Мы не будем рассматривать атрибуты отдельных элементов – все они содержатся в [спецификации HTML 4.01](#).

Единственное, на что нужно обратить внимание – случаи, когда название атрибута содержит дефис (например, `accept-charset`, `http-equiv`) или состоит из нескольких значимых слов. О правилах именования таких свойств объекта (ссылки на элемент) мы будем говорить позже.

Кроме атрибутов HTML элементов у ссылок на эти элементы есть уникальные свойства и методы, которые зависят от типа элемента.

Свойства элемента `form`

Название	Что возвращает	Что означает
<code>elements</code>	HTML-коллекция	HTML-коллекция элементов формы Только чтение
<code>length</code>	число	Количество элементов формы – длина массива <code>elements</code> Только чтение
<code>name</code> <code>acceptCharset</code> <code>action</code> <code>enctype</code> <code>method</code> <code>target</code>	строка или <code>undefined</code>	Доступ к значениям HTML атрибутов формы

Методы элемента `form`

Название	Аргументы	Что делает
<code>submit()</code>	–	Отправляет форму
<code>reset()</code>	–	Возвращает форму в первоначальное состояние

Главная особенность методов формы – они не вызывают соответствующих событий. Это означает, что метод `submit()` не создаст события `onsubmit`. Однако, это не вызывает осложнений – если мы посылаем форму с помощью вызова метода, то перед отправкой можем проделать все необходимые операции.

Элементы форм

Коллекцию формы `elements` мы будем рассматривать более подробно.

Доступ к любому элементу формы может быть получен с помощью его индекса. Индекс определяется положением этого элемента в коде страницы.

```
<form name="search" method="get">
<input name="word">
...
var i = document.search.elements[0];
```

Большинство элементов форм имеют атрибут `name`, который используется серверными скриптами, принимающими форму. Это позволяет обращаться к ним с использованием значения этого атрибута по аналогии с формами.

```
<form name="search" method="get">
<input name="word">
...
var i = document.search.elements.i;
```



И ровно так же все значения атрибутов `name` элементов форм попадают в перечень свойств объекта-ссылки на эту форму. Именно по этой причине, как и для форм, следует избегать употребления в качестве значения атрибута `name` элементов формы названий свойств и методов форм.

Количество элементов формы может быть получено как с помощью свойства формы `length`, так и с помощью свойства коллекции `elements.length`.

Классификация элементов форм

Все элементы форм можно разделить на несколько групп по их назначению.

Элементы для ввода информации

Элемент	Что означает	JS свойства	Методы	HTML свойства
<code><input type="text"></code> 	Текстовое однострочное поле	<code>form</code> <code>defaultValue</code>	<code>select()</code> <code>blur()</code> <code>focus()</code>	<code>type</code> <code>name</code> <code>value</code> <code>disabled</code> <code>readOnly</code> <code>tabIndex</code> <code>maxLength</code> <code>size</code> <code>accessKey</code>
<code><input type="password"></code> 	Поле для ввода пароля	<code>form</code> <code>defaultValue</code>	<code>select()</code> <code>blur()</code> <code>focus()</code>	<code>type</code> <code>name</code> <code>value</code> <code>disabled</code>




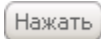
				readOnly tabIndex maxLength size accessKey
<code><textarea></textarea></code>	Поле для ввода многострочного текста	form defaultValue type value	select() blur() focus()	name disabled readOnly tabIndex rows cols accessKey

Элементы выбора

Элемент	Что означает	JS свойства	Методы	HTML свойства
<code><input type="radio"></code> <input checked="" type="radio"/> Выбрать <input type="radio"/> Выбрать	Выбор «один из многих» Радиокнопки	form defaultChecked	click() blur() focus()	type name value disabled checked tabIndex accessKey
<code><input type="checkbox"></code> <input checked="" type="checkbox"/> Выбрать <input type="checkbox"/> Выбрать	Выбор «многие из многих» Чекбоксы	form defaultChecked	click() blur() focus()	type name value disabled checked tabIndex accessKey
<code><select></select></code> <div>Выбрать ▼</div>	Выбор «один из многих» Выпадающий список	form options length selectedIndex value	add() remove() blur() focus()	name disabled tabIndex multiple size
<code><select multiple></select></code> <div> <div>Выбрать</div> <div>Выбрать</div> <div>Выбрать</div> <div>Выбрать</div> <div>Выбрать</div> <div>Выбрать</div> </div>	Выбор «многие из многих» Развернутый список	type		
<code><input type="file"></code> <div> <input type="text"/> <input type="button" value="Обзор"/> </div>	Поле для выбора файла Выглядит по-разному в различных браузерах	form defaultValue	click() blur() focus()	type name value disabled

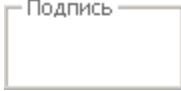
				tabIndex accept size accessKey
--	--	--	--	---

Кнопки

Элемент	Что означает	JS свойства	Методы	HTML свойства
<input type="submit"/> <button type="submit"></button> 	Отправка формы	form	click() blur() focus()	type name value disabled tabIndex size accessKey
<input type="image"/> 	Отправка формы	form	click() blur() focus()	type name value disabled tabIndex useMap src alt accessKey
<input type="reset"/> <button type="reset"></button> 	Возврат формы к первоначальным установкам	form	click() blur() focus()	type name value disabled tabIndex size accessKey
<input type="button"/> <button type="button"></button> 	Запуск скрипта	form	click() blur() focus()	type name value disabled tabIndex type size accessKey

Вспомогательные элементы

Элемент	Что означает	JS свойства	Методы	HTML свойства
<input type="hidden">	Скрытое поле для хранения служебной информации	form	–	type value

<label> </label>	Передача фокуса элементу формы	form	–	htmlFor accessKey
<fieldset> </fieldset> <legend> </legend>	Группировка элементов формы Подпись к сгруппированным элементам формы	form	–	accessKey (legend)
				

В выпадающем списке находятся несколько элементов, которые формируют отдельные строки и группы строк.

Отдельные элементы выпадающих списков

Элемент	Что означает	JS свойства	Методы	HTML свойства
<option> </option>	Отдельный элемент списка	form text defaultSelected index	–	value selected disabled label
<optgroup> </optgroup>	Группировка элементов списка	–	–	disabled label

В этом списке можно видеть как уже знакомые атрибуты HTML-элементов, так и новые, появившиеся только в JavaScript. Как уже отмечалось, значения атрибутов могут быть получены как значения свойств.

Как значения атрибутов, так и свойства могут быть предназначены только для чтения. Это означает, что получить мы их можем, а изменить нет.

Значения некоторых свойств и атрибутов

Отдельные атрибуты и свойства будут полезны при работе с элементами формы. С их помощью мы будем управлять поведением этих элементов, работать с их значениями, модифицировать элементы и добавлять их.

Отдельные атрибуты и свойства элементов форм

Название	Значение	Что означает
form	ссылка на элемент	Ссылка на форму, в которой находится элемент Только для чтения
checked	true false	Текущее состояние радиокнопок и чекбоксов Выделен – true, не выделен – false
defaultChecked	true false	Начальное состояние радиокнопок и чекбоксов Выделен – true, не выделен – false При сбросе формы влияет на выделение
value	строка	Текущее значение этого атрибута элемента или содержание textarea
defaultValue	строка	Начальное значение этого атрибута элемента или содержания textarea

		Имеется только у элементов текстового ввода При сбросе формы содержание элемента возвращается к этому значению
type	строка	Для элементов input, button соответствует значению атрибута type Для textarea вернет textarea Для выпадающего списка вернет select-one Для выпадающего списка с атрибутом multiple вернет select-multiple
options	HTML-коллекция	Коллекция элементов option внутри select Только для чтения
length	число	Количество элементов option внутри select Равносильно options.length Только для чтения
text	строка	Текст внутри элемента option
selected	true false	Текущее состояние элемента option Выделен – true, не выделен – false
defaultSelected	true false	Начальное состояние элемента option Выделен – true, не выделен – false При сбросе формы влияет на выделение
selectedIndex	число	Индекс выделенного элемента option в коллекции options Если не выделен ни один option, вернет -1 Для списков с мультिवыбором вернет индекс первого из выделенных элементов
index	число	Индекс элемента option в коллекции options Только для чтения

Методы элементов форм и события

Для элементов форм можно вызвать общие и уникальные методы

Методы элементов форм

Название	Элементы	Аргументы	Что делает
focus()	Большинство элементов	–	Передает фокус элементу Для текстовых полей помещает курсор ввода в начало поля
blur()	Большинство элементов	–	Элемент теряет фокус
select()	Элементы ввода	–	Передает фокус и выделяет содержание
click()	Элементы выбора, кроме выпадающего списка, кнопки	–	Имитирует клик мышкой на элементе

Большинство видимых элементов формы имеют методы `focus()`, `blur()`, которые устанавливают или убирают фокус с элемента. На невидимый, скрытый элемент, или элемент с атрибутом `disabled` передать фокус нельзя.

Эти методы связаны с событиями, которые возникают при их вызове. Так, при вызове метода `select()` для элемента ввода возникнут события `onselect` и `onfocus`.

Для элементов ввода формы нам будут полезны события клавиатуры `onkeydown`, `onkeypress`. С их помощью можно отслеживать действия пользователя при заполнении формы.

Для некоторых элементов полезно будет использовать событие `onchange`, которое позволяет отследить изменения в этом поле. Однако следует помнить, что это событие возникает только после потери фокуса этим элементом. Также нужно знать, что изменение содержания `textarea` или атрибута `value` с помощью JavaScript не вызывает это событие.

Работа с выпадающими списками

В первых версиях DOM добавление и изменение элементов было доступно только в элементе `select`. Это было связано со сложностями в работе анализаторов страниц при их перерисовке. Выпадающий список же занимал уже отведенное ему место и новые элементы не влияли на общее расположение элементов.

Для добавления новых элементов `option` был введен класс-конструктор `Option`. Создать новый элемент `option` можно было с помощью вызова этого класса-конструктора.

```
var o = new Option();
```

Класс-конструктор `Option`

Название	Аргументы	Что делает
<code>Option(text, value, defaultSelected, selected)</code>	<code>text</code> – строка текст внутри создаваемого элемента <code>value</code> – строка значение атрибута <code>value</code> <code>defaultSelected</code> – логическое значение начальное выделение <code>selected</code> – логическое значение текущее выделение	Создает новый элемент <code>option</code> и возвращает ссылку на него Все аргументы необязательны Их изменение возможно после создания в любой момент

После создания элемента его можно вставить в `select` с помощью метода `add()`.

Для удаления существующего элемента `option` применяется метод `remove()`

Методы элемента `select`

Название	Аргументы	Что делает
<code>add(new, old)</code>	<code>new</code> – ссылка на новый элемент <code>option</code> <code>old</code> – ссылка на элемент <code>option</code> , перед которым вставляется новый	Вставляет созданный элемент <code>option</code> перед указанным в качестве второго аргумента элементом <code>option</code> Для вставки созданного элемента в конец нужно указать для второго аргумента <code>null</code>
<code>remove(index)</code>	<code>index</code> – число	Удаляет элемент <code>option</code> с указанным индексом Если индекс меньше 0 или больше индекса последнего элемента, ничего не происходит

Это – первый случай, когда мы сталкиваемся с различной работой IE и остальных браузеров. Для IE в качестве второго аргумента нужно передавать не ссылку на него, а его индекс. Для вставки нового элемента в конец нужно просто опустить второй аргумент.

Для решения этой проблемы вспоминаем инструкцию исключения.

```
<form name='f'>
<select name='s'>
...
var o = new Option('Новый текст', 'a2', false, true);
var s = document.f.s;
try {
    s.add(o, s.options[2]); //вставляем перед option с индексом 2
} catch(e) {
    s.add(o, 2);
}
```

Изображения

Для нахождения изображений используется коллекция `document.images`. Как и остальные коллекции документа, она обладает той же особенностью – все изображения с атрибутом `name` доступны как одноименные свойства объекта `document`.

Тем не менее, поскольку атрибут `name` согласно стандартам могут иметь окна, формы и их элементы, изображения будут доступны по их индексу, который определяется их положением в коде страницы.

Для создания изображений был введен класс-конструктор `Image()`.

Класс-конструктор `Image`

Название	Аргументы	Что делает
<code>Image(w, h)</code>	<code>w</code> – число	Создает новый элемент <code>img</code> и возвращает ссылку на него
	ширина изображения	
	<code>h</code> – число	Все аргументы необязательны
	высота изображения	Их изменение возможно после создания в любой момент

Как и остальных ссылок на HTML-элементы атрибуты элемента доступны как свойства этого объекта.

Что очень важно, при задании свойству `src` значения адреса картинки браузер тут же посылает запрос на загрузку этого изображения. Поскольку в первых версиях DOM методов для вставки новых картинок не было, этот прием использовался для «предзагрузки» новых изображений, которые могли понадобиться пользователю в процессе работы со страницей.

Таблицы

Для модификации элементов таблиц были созданы уникальные методы и свойства, как для таблиц, так и для ее отдельных частей.

Таблица состоит из:

- заголовка `caption`
- секции `thead`
- секции `tfoot`
- секций `tbody`
- элементов `col`, `colgroup`

В свою очередь, секции содержат строки `tr` и ячейки `td`, `th`.

Если в коде пропущена секция `tbody`, а строки написаны непосредственно внутри `table`, браузер генерирует ее самостоятельно.

Свойства элемента `table`

Название	Значение	Что означает
<code>caption</code>	ссылка на элемент	Ссылка на элемент <code>caption</code> или <code>null</code> , если он отсутствует
<code>tHead</code>	ссылка на элемент	Ссылка на элемент <code>thead</code> или <code>null</code> , если он отсутствует
<code>tFoot</code>	ссылка на элемент	Ссылка на элемент <code>tfoot</code> или <code>null</code> , если он отсутствует
<code>tBodies</code>	HTML-коллекция	Массив элементов <code>tbody</code> внутри таблицы
<code>rows</code>	HTML-коллекция	Массив всех элементов <code>tr</code> внутри таблицы

Как и у любого массива, у коллекций есть свойство `length` – количество элементов этой коллекции.

Методы элемента `table`

Название	Аргументы	Что делает
<code>createCaption()</code>	–	Возвращает ссылку на элемент <code>caption</code> Если элемента не существует, то он создается и вставляется (пустой)
<code>deleteCaption()</code>	–	Удаляет элемент <code>caption</code> , если он существует
<code>createTHead()</code>	–	Возвращает ссылку на элемент <code>thead</code> Если элемента не существует, то он создается и вставляется (пустой)
<code>deleteTHead()</code>	–	Удаляет элемент <code>thead</code> , если он существует
<code>createTFoot()</code>	–	Возвращает ссылку на элемент <code>tfoot</code> Если элемента не существует, то он создается и вставляется (пустой)
<code>deleteTFoot()</code>	–	Удаляет элемент <code>tfoot</code> , если он существует
<code>insertRow(index)</code>	<code>index</code> – число	Вставляет пустую строку с заданным индексом

	индекс новой строки	Если индекс равен количеству строк в таблице, новая строка вставляется в ее конец Если таблица пустая, перед вставкой строки создается и вставляется секция <code>tbody</code> , в которую уже и будет вставлена строка
<code>deleteRow(index)</code>	<code>index</code> – число индекс строки в массиве <code>rows</code>	Удаляет строку с заданным индексом Индекс определяется положением в коде

Для работы со строками только внутри секций таблицы существуют аналогичные свойства и методы для этих секций.

Свойства секций `thead`, `tfoot`, `tbody`

Название	Значение	Что означает
<code>rows</code>	HTML-коллекция	Массив всех элементов <code>tr</code> внутри этой секции таблицы

Методы секций `thead`, `tfoot`, `tbody`

Название	Аргументы	Что делает
<code>insertRow(index)</code>	<code>index</code> – число индекс новой строки	Вставляет пустую строку с заданным индексом в секцию Если индекс равен количеству строк в секции, новая строка вставляется в ее конец
<code>deleteRow(index)</code>	<code>index</code> – число индекс строки в массиве <code>rows</code>	Удаляет строку с заданным индексом Индекс определяется положением в коде

Для работы с ячейками таблицы у строк есть свои свойства и методы.

Свойства элементов `tr`

Название	Значение	Что означает
<code>rowIndex</code>	число	Индекс строки в массиве <code>rows</code> таблицы
<code>sectionRowIndex</code>	число	Индекс строки в массиве <code>rows</code> секции таблицы, внутри которой размещается строка
<code>cells</code>	HTML-коллекция	Массив всех элементов <code>td</code> , <code>th</code> внутри строки

Методы элементов `tr`

Название	Аргументы	Что делает
<code>insertCell(index)</code>	<code>index</code> – число индекс новой ячейки	Вставляет пустую ячейку с заданным индексом в строку

		Если индекс равен количеству ячеек в секции, новая ячейка вставляется в ее конец
deleteCell(index)	index – число индекс ячейки в массиве cells	Удаляет ячейку с заданным индексом Индекс определяется положением в коде

Печальный факт – при вызове метода insertCell() всегда создается элемент td.

Для записи содержания в таблицу будем использовать пока не изученное свойство HTML-элементов innerHTML.

Объектная модель документа (W3C DOM)

В DOM Level 2 представление документа расширено – теперь все элементы документа представляют собой дерево, в основе которого лежит объект Document. Каждый элемент этого дерева называется узлом (Node). Узлом дерева является все, что встречается на странице, включая текст.

Тип узлов

Интерфейсы

Каждый узел, в зависимости от его типа, обладает набором свойств и методов. Ряд из этих свойств и методов являются универсальными, но часть представляет собой уникальный набор.

Все узлы дерева имеют общий так называемый интерфейс Node – набор заранее реализованных свойств и методов. Однако, в зависимости от типа узла для каждого из них реализован свой дополнительный интерфейс.

Ранее мы рассматривали классы в базовом языке JavaScript – для каждого из них существует свой тип данных. В отличие от классов, интерфейсы не создают свой тип данных. Они просто реализуют набор свойств и методов для отдельных видов элементов.

Кроме того, добавление отдельных интерфейсов для элементов приводит просто к увеличению базового набора свойств и методов, не затрагивая сущность самих элементов.

Типы узлов

Название интерфейса	nodeType	Что означает
Element	1	Любой html-элемент на странице
Text	3	Текст
Document	9	Объект Document
Comment	8	Комментарий
DocumentFragment	11	Несколько узлов, объединенных в один фрагмент (используется при создании и добавлении узлов в документ)
Attr	2	Набор атрибутов html-элемента

Получив ссылку на какой-либо узел DOM-структуры, можно всегда узнать его тип с помощью свойства nodeType.

Работа с атрибутами элементов предполагалась в виде отдельного интерфейса. Связано это с тем, что атрибуты могут быть просто указаны непосредственно в коде, а могут иметь некие значения по умолчанию (встроенные стили браузера). Однако атрибуты не присутствуют в коде страницы в виде отдельных узлов, как текст или сами элементы.

По причине сложности работы с атрибутами по предложенной схеме, а также в связи с разной реализацией этого интерфейса разными браузерами, на практике используют доступ к атрибутам через свойства (как это было рассмотрено ранее) и методы самих элементов.

Ниже мы рассмотрим интерфейс Node и его подынтерфейсы, реализованные для отдельных типов узлов.

Иерархия и взаимоотношение узлов

Перед тем, как рассматривать отдельные типы узлов, нужно понять иерархию древовидной структуры.

Родительский и дочерний элементы

Узел, в который непосредственно вложен другой, называется его **родителем** или родительским. В свою очередь, непосредственно вложенный узел является **дочерним** по отношению к своему родителю.

```
<div>
  <p class="top">
    <a></a>
  </p>
  <p></p>
</div>
```

В описанной структуре элемент div является родительским по отношению к обоим параграфам. Первый параграф – родитель гиперссылки.

Дочерним элементом элемента div будут оба параграфа, а гиперссылка будет дочерним элементом первого параграфа с классом top.

Предки и потомки

Все внутренние элементы, независимо от глубины вложенности, называются **потомками**. И параграфы, и гиперссылка будут потомками элемента div.

В свою очередь, любой родитель, прародитель и т.д. для элемента будут являться его **предком**.

В примере элемент div будет предком для всех его потомков.

Соседи

Узлы, находящиеся на одном уровне и имеющие одного родителя, называются **соседями** или **братьями**. В нашем примере соседями будут параграфы.

Интерфейс Node

Свойства и методы этого интерфейса присущи всем узлам дерева документа.

Свойства интерфейса Node – иерархия узлов документа

Название свойства	Термин	Что означает
parentNode	родитель	Узел, в который непосредственно вложен данный узел
childNodes	дочерние узлы	Массив всех дочерних узлов
firstChild	первый дочерний узел	Первый элемент массива childNodes
lastChild	последний дочерний узел	Последний элемент массива childNodes
nextSibling	следующий сосед	Соседний узел, находящийся в коде после данного узла
previousSibling	предыдущий сосед	Соседний узел, находящийся в коде перед данным узлом
ownerDocument	Document	Ссылка на документ, в котором находится узел

Свойства интерфейса Node – тип узла

Название свойства	Что содержит	Что означает
nodeName	имя узла	Для элементов возвращает имя тэга Для остальных типов узлов – общее значение для этого типа
nodeType	тип узла	Для универсальности возвращает цифру – код типа узла
nodeValue	содержание узла	Для текстового узла возвращает текст – содержание узла Для остальных типов узлов – null
attributes	массив атрибутов узла	В IE реализован свой уникальный массив По этой причине не используется

Методы интерфейса Node

Название	Аргументы	Что делает
hasChildNodes()	–	Возвращает true, если у узла есть дочерние элементы
hasAttributes()	–	Возвращает true, если у узла есть атрибуты
normalize()	–	Объединяет смежные текстовые узлы (соседей) и удаляет пустые текстовые узлы
cloneNode(b)	b – логическое значение	Клонирует узел с его атрибутами и возвращает новый узел Если аргумент равен true, то узел копируется вместе с его содержимым
appendChild(e)	e – ссылка на	Вставляет узел в конец дочерних

	вставляемый узел	элементов
		<p>После вставки свойство lastChild узла, в который вставляем, ссылается на вставленный узел</p> <p>Если вставляемый узел уже был в дереве, сначала происходит его удаление из прежнего места</p> <p>Возвращает ссылку на вставленный узел</p>
insertBefore(e, p)	<p>e – ссылка на вставляемый узел</p> <p>p – ссылка на узел, перед которым вставляется первый</p>	<p>Вставляет узел перед узлом, указанным в качестве второго аргумента</p> <p>Если вставляемый узел уже был в дереве, сначала происходит его удаление из прежнего места</p> <p>Возвращает ссылку на вставленный узел</p>
replaceChild(e, p)	<p>e – ссылка на вставляемый узел</p> <p>p – ссылка на узел, который заменяется первым</p>	<p>Заменяет узел, указанный в качестве второго аргумента, на узел, указанный в качестве первого аргумента</p> <p>Возвращает удаленный узел</p>
removeChild(e)	e – ссылка на удаляемый узел	<p>Удаляет узел, указанный в качестве аргумента</p> <p>Возвращает ссылку на удаленный узел</p>

Интерфейс Element

Помимо указанных универсальных для всех узлов свойств и методов, для узлов, представляющих собой html-элементы, есть свои уникальные особенности.

Свойства интерфейса Element

Название свойства	Что содержит	Что означает
tagName	имя узла	Возвращает имя тэга Свойство для html-элементов идентично свойству nodeName

Методы интерфейса Element

Название	Аргументы	Что делает
getElementsByTagName(t)	t – название тэга	Возвращает массив всех потомков с указанным именем тэга
hasAttribute(a)	a – название атрибута	Возвращает true, если у узла есть установленный атрибут с указанным названием
getAttribute(a)	a – название атрибута	Возвращает значение атрибута с указанным названием
removeAttribute(a)	a – название атрибута	Удаляет атрибут с указанным названием
setAttribute(a, v)	a – название атрибута v – значение атрибута	Устанавливает новое значение атрибута с указанным названием

Интерфейс Document

Помимо указанных универсальных для всех узлов свойств и методов, для узлов, представляющих собой html-элементы, есть свои уникальные особенности.

Свойства интерфейса Document

Название свойства	Что содержит	Что означает
defaultView	ссылка на элемент	Ссылка на объект Window, в котором находится Document Для IE реализовано альтернативное свойство parentWindow
documentElement	ссылка на элемент	Ссылка на корневой элемент В HTML-страницах это – html
body	ссылка на элемент	Ссылка на элемент body
styleSheets	массив	Массив всех таблиц стилей, подключенных к документу
doctype	ссылка на элемент	Узел DocumentType – декларация документа В IE не поддерживается

Методы интерфейса Document

Название	Аргументы	Что делает
createComment(s)	s – строка	Создает и возвращает ссылку на новый узел Comment, содержащий указанную строку
createDocumentFragment()	–	Создает и возвращает ссылку на новый пустой узел DocumentFragment
createElement(t)	t – название тэга	Создает и возвращает ссылку на новый узел Element с указанным именем тэга
createTextNode(s)	s – строка	Создает и возвращает ссылку на новый узел Text, содержащий указанный текст
getElementById(i)	i – значение атрибута id	Возвращает элемент, находящийся в документе и имеющий указанное значение атрибута id Если такой элемент в документе отсутствует, возвращает null

Нестандартные свойства и методы HTML-элементов

В разное время появились нестандартные свойства, тем не менее, получившие поддержку практически во всех браузерах. Большая часть этих свойств была реализована в IE. Эти свойства будут включены в последующие стандарты DOM.

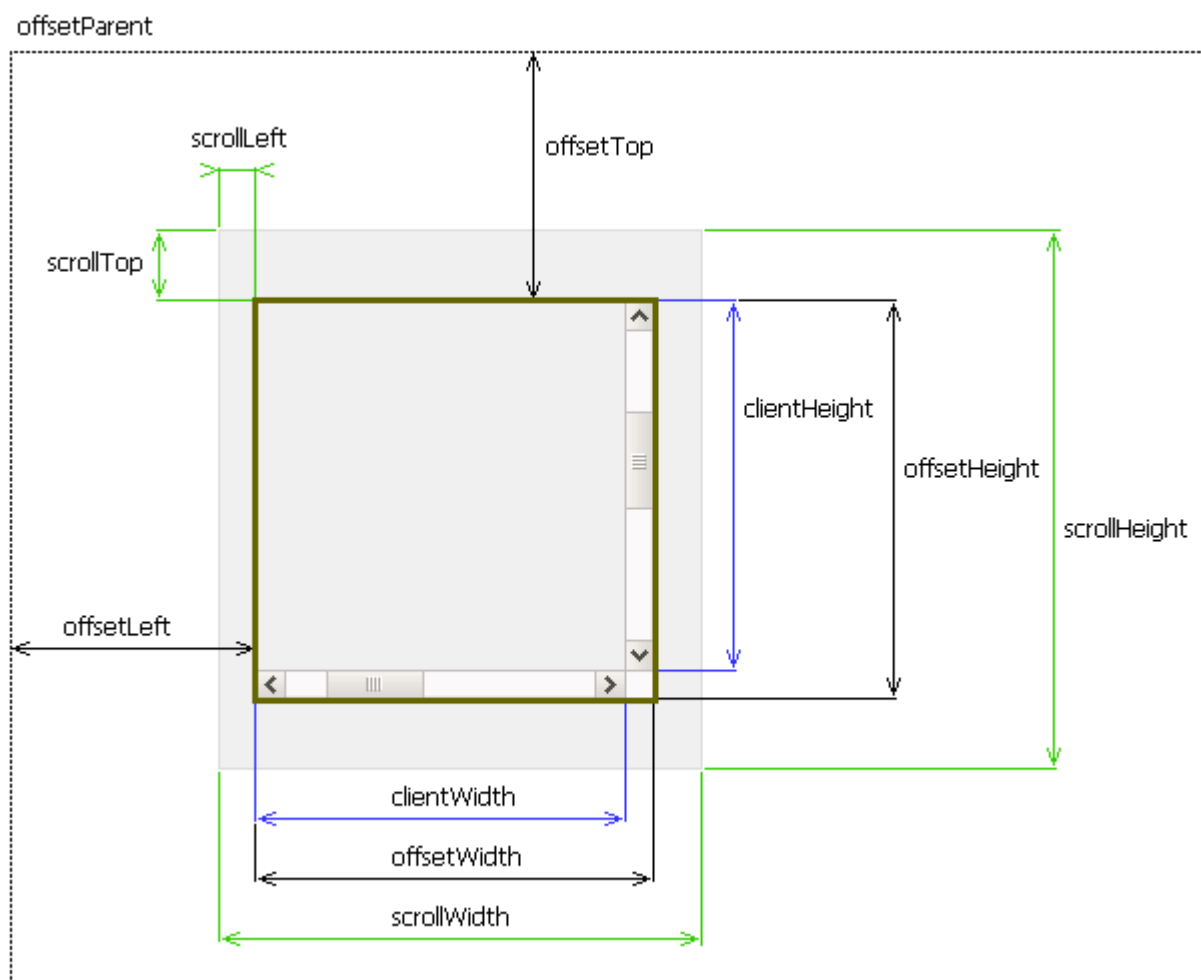
Большинство предназначено для получения информации о размерах и положении элемента.

Нестандартные свойства HTML-элементов

Название свойства	Что содержит	Что означает
innerHTML	строка HTML-текста	Доступная для чтения и записи строка, определяющая html-содержание элемента После изменения содержания элемента происходит так называемая сериализация – анализ и включения содержания в дерево документа
offsetWidth	число	Ширина элемента, включающая внутренние отступы (padding) и рамки (border), но не включающая внешние отступы (margin) Если элемент имеет вертикальную прокрутку, ширина полосы включается
offsetHeight	число	Высота элемента, включающая внутренние отступы (padding) и рамки (border), но не включающая внешние отступы (margin) Если элемент имеет горизонтальную прокрутку, ширина полосы включается
offsetLeft	число	Вертикальная координата левого верхнего угла элемента относительно специального контейнера offsetParent Внешние отступы не учитываются
offsetTop	число	Горизонтальная координата левого верхнего угла элемента относительно специального контейнера offsetParent Внешние отступы не учитываются
offsetParent	число	Контейнер, определяющий систему координат, относительно которой определяются offsetLeft и offsetTop Этот контейнер появляется динамически в результате правил CSS (позиционирование, смещение и т.д.) и особенностей отдельных элементов (строки, ячейки таблицы и т.д.)
scrollWidth	число	Полная ширина элемента без внешних отступов Используется для элементов с прокруткой (overflow) Если появляется горизонтальная прокрутка, вернет значение,

		отличающееся от <code>offsetWidth</code>
<code>scrollHeight</code>	число	<p>Полная высота элемента без внешних отступов</p> <p>Используется для элементов с прокруткой (<code>overflow</code>)</p> <p>Если появляется вертикальная прокрутка, вернет значение, отличающееся от <code>offsetHeight</code></p>
<code>scrollLeft</code>	число	Количество пикселей, на которое содержимое элемента было прокручено влево
<code>scrollTop</code>	число	Количество пикселей, на которое содержимое элемента было прокручено вверх
<code>clientWidth</code>	число	<p>Ширина клиентской части элемента</p> <p>Имеет значение для элементов с горизонтальной прокруткой</p>
<code>clientHeight</code>	число	<p>Высота клиентской части элемента</p> <p>Имеет значение для элементов с вертикальной прокруткой</p>

Ниже показаны значения этих свойств.



Свойства offset

Поддержка этих свойств различается в различных браузерах в режиме quirksmode.

Следует помнить, что offsetParent определяется по своим критериям и означает далеко не одно и то же, что parentNode.

Для определения отступов элемента от левого верхнего угла страницы следует пройти по всей цепочке элементов offsetParent. К сожалению, если в этой цепочке встретится элемент с "position:relative", то возникнет ошибка в определении этих отступов. По этой причине полагаться на этот метод очень сложно.

Если запросить у скрытого элемента ("display:none", "visibility:hidden") свойства offsetWidth, offsetHeight, они вернут нам 0.

Нестандартные методы HTML-элементов

Название	Аргументы	Что делает
scrollIntoView(b)	b – логическое значение (необязательный аргумент)	Прокручивает страницу таким образом, чтобы элемент был прижат к верхней или к нижней части окна Если аргумент пропущен или равен true, элемент прокручивается к верхней части окна Для false прокручивается к нижней части окна

Элемент iframe

Мы уже познакомились с коллекцией frames глобального объекта. В случае включения на страницу элемента iframe с атрибутом name мы легко можем использовать эту коллекцию для получения доступа к глобальному объекту, который формируется этим оконным элементом.

```
<iframe name="some"></iframe>
...
var some = frames['some']; //глобальный объект этого окна
some.document.body; //доступ к элементам загруженной страницы
```

Однако в случае работы с iframe как с html-элементом страницы для получения доступа к содержимому документа, загруженному в этот iframe следует использовать специальное свойство.

Свойство элемента iframe

Название свойства	Что содержит	Что означает
contentDocument	ссылка на объект	Ссылка на объект Document загруженный в iframe страницы

```
<iframe name="some" id="some"></iframe>
...
var some = document.getElementById('some'); //просто элемент главной страницы
some.contentDocument.body; //доступ к элементам загруженной страницы
```

Работа со стилями документа

Коллекция таблиц стилей StyleSheets

У Document есть коллекция всех таблиц стилей, подключенных с помощью элементов link и style. Следует помнить, что эти элементы могут находиться только в разделе head.

Доступ к этому массиву мы получаем с помощью свойства styleSheets.

```
var styles = document.styleSheets;
```

Элементы этого массива ведут себя, как обычные DOM-элементы. То есть можно их удалить, используя рассмотренные выше методы.

Свойства коллекции styleSheets

Название свойства	Что содержит	Что означает
length	число	Количество таблиц стилей

Каждый элемент этого массива – отдельная таблица стилей. У элемента есть свойства и методы, с помощью которых осуществляется доступ к отдельным правилам.

Свойства элемента коллекции styleSheets

Название свойства	Что содержит	Что означает
cssRules	коллекция	Массив всех правил В IE используется свойство rules

Общее количество правил можно получить через свойство length.

```
var rules_number = document.styleSheets[0].cssRules.length;
```

Методы элемента коллекции styleSheets

Название	Аргументы	Что делает
insertRule(r, i)	r – описание правила (строка) i – индекс (число)	Вставляет правило на позицию с указанным индексом Описание правила вида: 'div{padding:10px;}'
deleteRule(i)	i – индекс (число)	Удаляет правило с указанным индексом

Однако, эти методы не поддерживаются в IE – он определяет свои.

Методы элемента коллекции styleSheets – IE

Название	Аргументы	Что делает
addRule(s, r, i)	s – селектор (строка) r – описание правила (строка) i – индекс (число,	Вставляет правило на позицию с указанным индексом Селектор вида: 'div'

	необязательный аргумент)	Описание правила вида: 'padding:10px;' Если не указать индекс, правило вставляется в конец таблицы стилей
removeRule(i)	i – индекс (число)	Удаляет правило с указанным индексом Если не указать индекс, удаляется первое правило

У отдельного правила есть по факту единственное кроссбраузерное свойство.

Свойства правила

Название свойства	Что содержит	Что означает
cssSelector	строка	Селекторы для этого правила

Работа с вложенными стилями элементов

У любого элемента есть возможность назначить правила CSS с помощью атрибута `style`. Поскольку мы уже отмечали, что с атрибутами HTML-элементов можно работать, как со свойствами объектов, то следует ожидать, что к содержанию атрибута `style` мы можем получить доступ, запросив это свойство.

Однако запись и чтение правил CSS из значения этого атрибута имеет особенность. Каждое правило CSS мы записываем, устанавливая его в некоторое значение следующим образом.

```
d.style.padding = '10px';
```

Установить несколько правил одновременно невозможно. Важно понимать, что работать с вложенными стилями следует только в особых ситуациях:

- когда мы размещаем элементы с необходимым оформлением на чужие страницы
- для создания динамических эффектов

В остальных случаях следует просто устанавливать атрибут `class` в нужное значение.

Причин несколько.

Во-первых, принято разделять оформление страниц от функциональности. Внешний вид страниц должен формироваться с помощью CSS и изображений.

Во-вторых, некоторые свойства имеют свои особенности. В частности, нельзя скрывать и показывать таблицы, устанавливая для них свойство `display` поочередно в `none` и `block`. Дело в том, что везде, кроме IE для таблиц существует свое значение `table`. Чем мучаться с кроссбраузерностью, проще создать класс, отвечающий за скрытие и оперировать им. Это касается и тех свойств, которые реализуются в IE с помощью фильтров (Alpha, AlphaImageLoader).

Особенности именования свойств-аналогов правил CSS и атрибутов HTML

Некоторые правила CSS, атрибуты HTML-элементов имеют в составе дефис или состоят из нескольких слов. В этом случае принято применять «верблюжью» нотацию.

```
border-top-width => borderTopWidth
```

Есть соглашение, по которому решается конфликт имен атрибутов и правил с зарезервированными словами в JavaScript. Для правил CSS перед ними добавляется `"css"`, перед названиями атрибутов добавляется `"html"`.

Таких случаев немного.

```
for => htmlFor  
float => cssFloat
```

Единственное исключение из принятых правил решения конфликтных ситуаций существует для атрибута `class`.

```
class => className
```

Получение текущего значения правила CSS

В реализации DOM 2 у объекта **document.defaultView** существует метод, позволяющий получить текущее значение правила CSS, учитывая псевдоэлементы. Это будет значение, применяемое в текущий момент.

```
p{font-size:20px;}
p:first-letter{font-size:40px;}
...
var x = document.getElementsByTagName('p')[0];
var s1 = document.defaultView.getComputedStyle(x, ':first-letter').fontSize;
//20px
var s2 = document.defaultView.getComputedStyle(x, null).fontSize; //40px
```

К сожалению, в IE отсутствует реализация объекта "document.defaultView". Взамен метода этого объекта есть свой метод **currentStyle**. Это свойство не позволяет учитывать псевдоэлементы.

```
var s = x.currentStyle.fontSize; //20px
```

Для кроссбраузерного получения текущего значения правила можно исправить положение с помощью определения метода глобального объекта для IE. При этом мы учитываем, что **document.defaultView** фактически представляет собой ссылку на объект **Window**.

```
p{font-size:20px;}
...
if (!window.getComputedStyle) {
    window.getComputedStyle = function(element, pseudoElement) {
        return element.currentStyle;
    }
}
var s = getComputedStyle(x, null).fontSize; //20px
```

На практике проще написать кроссбраузерную функцию, возвращающую значение правила.

```
function getStyle(element, style) {
    if (window.getComputedStyle) {
        return getComputedStyle(element, null)[style];
    } else {
        return element.currentStyle[style];
    }
}
```

Модель событий в объектной модели документа

В отличие от рассмотренного способа регистрации обработчиков событий непосредственно в атрибутах HTML-элементов, модель обработки событий в DOM 2 предполагает гораздо более сложную схему. Эта модель часто называется «модель событий W3C DOM». К сожалению, в IE реализована своя схема возникновения и обработки событий.

В итоге в настоящий момент существует параллельно 3 модели реализации обработки событий

- Исходная модель
- Модель DOM 2
- Модель IE

Мы рассматривали исходную модель – регистрацию обработчиков событий в качестве значений атрибутов. Ниже будут представлены 2 оставшихся модели, так как особенности IE необходимо учитывать при написании кроссбраузерного кода.

Модель событий W3C DOM

Весь событийный процесс можно разделить на 3 фазы.

Фаза перехвата или распространения

При наступлении события оно возникает в объекте Document и распространяется вниз по дереву до узла, в котором произошло.

На этом этапе любой узел-предок может перехватить распространение события и с помощью функции-обработчика выполнить некие действия, вплоть до отмены дальнейшего распространения этого события вниз.

Некоторые события возникают непосредственно в объекте Window и никуда не распространяются. О таких случаях мы поговорим ниже в разделе «Отличия целевых элементов для некоторых событий».

Фаза регистрации

На этом этапе событие регистрируется в узле, в котором это событие произошло.

Этот этап модели событий аналогичен тому, что было и в более ранней модели событий, где событие регистрировалось непосредственно в элементе с помощью атрибутов.

Фаза всплытия

Событие «всплывает» обратно к объекту Document.

На этом этапе событие может «прослушать» любой узел-предок и выполнить свои действия.

Не все события имеют фазу «всплытия» – некоторые события интересны только тем узлам, в которых они и возникают. Таких событий немного – это потеря и получение фокуса узлами документа (окно, гиперссылки, элементы форм), отправка и очистка формы, ошибки при загрузке изображений.

При выполнении кода функции-обработчика ссылка на произошедшее событие передается в эту функцию в качестве первого аргумента.

Такая сложная модель дает несравненно большую гибкость по сравнению со старой схемой.

Во-первых, разные функции могут быть назначены для обработки одного и того же события.

Во-вторых, на этапе всплытия событие может быть «прослушано» и обработано любым из предков узла, в котором оно произошло.

Модель событий IE

В IE, начиная с 5-й версии, реализована иная модель событий.

Фаза распространения отсутствует

Фаза распространения в этой модели отсутствует. Остальные фазы идентичны - событие регистрируется в узле и всплывает.

Событие в виде свойства глобального объекта

Ссылка на событие не передается в функцию-обработчик, а существует в виде свойства глобального объекта window.event. Такое поведение возможно в связи с тем, что код JavaScript всегда выполняется последовательно. Поэтому в один момент времени может обрабатываться только одно событие.

Есть и другие отличия (мы про них узнаем позже), но указанные являются основными.

Работа с обработчиками событий

Регистрация

Для регистрации обработчиков события и ее отмены существуют методы, присущие всем HTML-элементам. Пусть переменная `x` будет представлять ссылку на объект – HTML-элемент, и существует функция `handler()`.

```
var x = document.getElementById('x');  
function handler(e){  
    //здесь исполняемый код  
}
```

Методы регистрации обработчиков событий

Модель DOM 2	Модель IE
<code>x.addEventListener('click', handler, false)</code>	<code>x.attachEvent('onclick', handler)</code>

Отметим основные различия этих методов.

Первое состоит в передаваемом названии события – в модели DOM 2 приставка “on” отсутствует.

Второе отличие – последний аргумент в модели DOM 2. Поскольку в IE фаза распространения отсутствует, на практике приходится жертвовать этим этапом и повсеместно можно наблюдать регистрацию обработчиков в модели DOM 2 с параметром `false`.

Так же, как и ранее, можно в качестве обработчика указывать как имя функции, так и безымянную функцию `function(){}.`

Отмена регистрации

Иногда возникает необходимость отменить обработку события. В этом случае мы используем соответствующие методы с передачей им тех же самых атрибутов, что и при регистрации события.

Методы отмены регистрации обработчиков событий

Модель DOM 2	Модель IE
<code>x.removeEventListener('click', handler, false)</code>	<code>x.detachEvent('onclick', handler)</code>

Ключевое слово `this` в обработчике события

В модели событий DOM 2 ключевое слово `this` в обработчике события ссылается на объект, в котором зарегистрирован обработчик события. Однако в модели событий IE этот обработчик выполняется в контексте глобального объекта `Window`. По этой причине для написания кроссбраузерного кода это слово, к сожалению, мы использовать не будем.

Получение ссылки на событие

Как уже отмечалось выше, в модели событий DOM 2 ссылка на событие передается в качестве аргумента обработчику события. В модели IE ссылка на событие содержится в свойстве глобального объекта `window.event`. По этой причине в коде обработчика события можно кроссбраузерно получить ссылку на событие следующим образом:

```
function handler(e){
    e = e || event;
    //далее исполняемый код
}
```

В результате выполнения первой строчки кода возможны 2 варианта.

- Ссылка на событие находится в локальной переменной `e`. Тогда переменной присваивается ее же значение.
- Ссылка на событие содержится в свойстве глобального объекта. Тогда вычисление значения выражения в правой части даст значение этого свойства.

Вместо `"event"` можно было бы написать `"window.event"`, однако это не имеет в данном случае принципиального значения, поскольку обработчик события выполняется для модели IE в контексте глобального объекта.

Отмена действий по умолчанию

В исходной модели мы могли отменить выполнение некоторых действий по умолчанию, вернув из их обработчика `false`. Например, можно было предотвратить загрузку документа при клике на ссылку, отправку и сброс формы, печать символов при нажатии клавиш.

Для выполнения такого же действия в рассматриваемых моделях есть свои методы и свойства.

```
function handler(e){
    e = e || event;
    //далее исполняемый код
}
```

Свойства и методы для отмены действий по умолчанию

Модель DOM 2	Модель IE
<code>e.preventDefault()</code>	<code>e.returnValue = false</code>

Можно отметить, что присвоение свойству `returnValue` значение `false` равносильно возврату `false` из обработчика события, что уже было рассмотрено в исходной модели событий.

Кроссбраузерные свойства события

После получения ссылки на событие можно обратиться к его свойствам, в которых содержится информация. Мы не будем брать в расчет те свойства, которые реализованы только для одной модели событий.

Рассмотрим только те свойства, значения которых могут быть в обеих моделях.

Свойства события

Название	Источник	Значение	Что означает
type	любой	Строка – название события	Тип произошедшего события
target (DOM) srcElement (IE)	любой	Ссылка на HTML-элемент	Ссылка на элемент, в котором событие произошло
button	мышь	Число	Число, указывающее на клавишу мыши, нажатую в момент возникновения события В разных браузерах это число может отличаться для всех клавиш, кроме левой
altKey	мышь	Логическое значение	Если в момент события была нажата клавиша Alt, вернется true
shiftKey	мышь	Логическое значение	Если в момент события была нажата клавиша Shift, вернется true
ctrlKey	мышь	Логическое значение	Если в момент события была нажата клавиша Ctrl, вернется true
clientX	мышь	Число	Координата по оси X относительно левого верхнего левого угла экрана до места, где произошло событие
clientY	мышь	Число	Координата по оси Y относительно левого верхнего левого угла экрана до места, где произошло событие
keyCode	клавиатура		ASCII код клавиши

Получение ссылки на элемент, в котором произошло событие

Поскольку контекст выполнения функции-обработчика события разный в модели DOM 2 и IE, то кроссбраузерно определять элемент, в котором произошло это событие, нужно следующим образом.

```
function handler(e){
    e = e || event;
    var target = e.target || e.srcElement;
    //далее исполняемый код
}
```

Точно так же, как и для получения ссылки на событие мы используем логическое «ИЛИ». В результате для локальной переменной target мы получаем значение, одинаковое для различных моделей событий.

Отличия целевых элементов для некоторых событий по сравнению с базовой моделью

В таблице событий, рассмотренной в 6-м модуле, ряд событий был связан с элементом <body>. Для обработки этих событий следовало написать для этого элемента соответствующий атрибут. В модели DOM 2 эти события перешли к объектам Window и Document. Нетрудно заметить, что к объекту Document перешли события клавиатуры, которые до сих пор не описаны в стандарте.

Список событий, перешедших от <body> к Window

Название	Элемент	Что означает
onfocus	Window	Окно получило фокус ввода
onblur	Window	Окно потеряло фокус ввода
onload	Window	Загрузка документа завершена
onunload	Window	Содержимое окна выгружается
onerror	Window	Ошибка в окне (в том числе и ошибки JavaScript)
onresize	Window	Изменение размеров окна
onscroll	Window	Внутреннее содержание окна было прокручено с помощью линейки или колеса мышки

Список событий, перешедших к Document

Название	Элемент	Что означает
onkeydown	Document	Клавиша нажата Для отмены возвращает false
onkeypress	Document	Клавиша нажата и отпущена Для отмены возвращает false
onkeyup	Document	Клавиша отпущена

Использование фазы всплытия событий

Представим ситуацию, в которой нам хотелось бы отследить событие “onclick” для всех гиперссылок на странице, которых на ней содержится несколько десятков. После загрузки документа нам пришлось бы найти коллекцию гиперссылок и в цикле назначить каждой из них обработчик этого события.

Вместо этого ресурсоемкого процесса мы можем использовать тот факт, что событие «всплывает» к объекту document, и перехватывать его именно в этом объекте. Затем следует просто определить, в каком именно типе элемента произошло событие и в интересующем нас случае обрабатывать его.

Предположим, что мы зарегистрировали в качестве обработчика события click функцию handler.

```
function handler(e) {  
  e = e || event;  
  var target = e.target || e.srcElement;  
  if (target.tagName == 'A') {  
    alert('Это - гиперссылка');  
  }  
}
```

Отмена всплытия события

Иногда нужно остановить всплывающее событие для того, чтобы сэкономить ресурсы на «прослушивание» события родительскими элементами, вплоть до объекта document.

Как и в случае отмены действия по умолчанию, в стандартной модели есть метод, в модели IE – свойство.

```
function handler(e){  
    e = e || event;  
    //далее исполняемый код  
}
```

Свойства и методы для отмены всплытия события

Модель DOM 2	Модель IE
<code>e.stopPropagation()</code>	<code>e.cancelBubble = true</code>

Кроссбраузерная работа с функциями-обработчиками событий

Для регистрации обработчиков событий и ее отмены мы не будем смотреть на имена и версии браузеров. Достаточно определять, поддерживается то или иное свойство или метод.

Посмотрим, как можно написать кроссбраузерную пользовательскую функцию, которая будет «навешивать» обработчик на событие.

```
function addEvent(element, event_name, handler){
    if (element.addEventListener) { //проверяем поддержку стандартного метода
        element.addEventListener(event_name, handler, false);
    } else if (element.attachEvent) { //проверяем поддержку модели IE
        element.attachEvent('on' + event_name, handler);
    } else { //для совсем древних браузеров
        element['on' + event_name] = handler;
    }
}
addEvent(document, 'click', function(e){...}); //вызываем созданную функцию
```

Ровно так же поступаем и с функциями, которые отменяют обработчики, всплытие событий, действия по умолчанию и т.д.

Библиотеки, которые часто используются в разных проектах (prototype, jquery и т.д.), фактически осуществляют разными способами ту же операцию – в них есть служебные функции, в реализации которых учитываются все нюансы браузеров по реализации JavaScript.