Часть 1 (Client- side)

Предисловие

Client- side JavaScript подразумевает возможности, доступные на стороне клиента (браузера). Это возможности чистого языка, и возможности браузера (а это, в свою очередь, возможности DOM (Document Object Model), и BOM (Browser Object Model)).
В Client- side JavaScript исполнение сценария происходит в контексте глобального объекта — объекта Window.

Client- side JavaScript

В Client- side JavaScript исполнение сценария происходит в контексте глобального объекта — объекта Window.
Как следует из названия, этот объект инкапсуллирует окно (/вкладку/фрейм) текущего документа. Для доступа к нему используется ключевое слово window. Это член объекта Window, содержащий ссылку на него самого. Обращаться к членам глобального объекта можно и уточненной записью (но, как обычно, это попросту избыточно).
Для доступа к URL текущего документа используется ссылка window.location. Сам URL храниться в этом объекте в виде строки. Присваивание пользовательского значения допустимо, и приведет к загрузке документа по указанному адресу (запись вида window.location = "http:// … ";).
Содержимое текущего окна инкапсуллируется объектом Document. Для ссылки на объект Document служит window.document.
Элементы документа хранятся в объектах Element.
Все эти объекты имеют характерные члены.
В том числе события (через которые подключается сценарий к документу).

Историческое

Исторически, применение JavaScript начиналось так. Изначально он предполагался для оснащения документов функциональностью приложений.
Приэтом обнаружилось следующее. Документы ранее не использовались таким путем — и чтобы быть эксплуатабельными в этом применении, они должны быть соответствующим образом спроектированы. И в интересах проектирования эксплуатабельных документов стали применять следующие подходы:
при одном веб- документы дожны быть больше похожи на документы, чем на приложения; функциональность сценариев предполагалась опциональной, и

доступность контента документа предполагалась не зависимой от доступности сценариев;
при другом подходе веб- документы должны быть больше похожи на приложения, чем на документы; функциональность сценариев при этом подходе полагалось безопциональной — соответственно, доступность контента документа могла быть зависимой от доступности сценариев; такие документы стало принято называть веб- приложениями;

Веб- документ, оснащенный функциональностью приложений, стали называть веб- приложением. Функциональность приложения обеспечивалась `JavaScript`.
В дальнейшем, для придания веб- документам функциональности приложений стали использовать и другие языки. И такие документы, по традиции, стали называть веб- приложениями.
Итак, формальное определение веб- приложения — это приложение, поставляющее свою функциональность через веб- интерфейс (то есть, пользовательский интерфейс, доступный из такого веб- документа).

Еще о подключении сценариев к документу

Подключаясь к документу (каким бы то ни было способом), сценарий получает доступ к нему (и прочим документам с того же домена, что и этот документ). Это правило доступа сценария к веб- документам называется политикой общего происхождения. Касательно нее необходимо помнить, что она регулирует только вышеописанное (источник самого сценария она никак не регулирует).
Подключение сценариев из событий может быть прописано как в `HTML`- коде (в значении атрибутов- событий), так и в `JavaScript`- коде (в объектах окна, документа, и элементов — для ссылки на обработчики событий используются соответствующие члены этих объектов, а сами обработчики событий в таком случае прописываются в виде функциональных выражений, присваиваемых этим ссылкам). Подключение сценариев из событий (каким- либо путем) принято называть регистрацией обработчиков событий. Запись подключения сценария из событий в `JavaScript`- коде имеет следующий вид:

```
ObjectName.EventName = function( … ) { … };
/* ObjectName принято называть адресатом события,
EventName – приемником или обработчиком события */
```

Подключение сценариев из кода самого веб- документа уже считается устаревшим, и давно не практикуется (что намекает о перспективе дальнейшей доступности таких способов подключения). Хотя, подключение сценария посредством псевдопротокола может и в перспективе оставаться доступным (это потому, что уже устоялась традиция применять псевдопротокол для подключения тривиальных сценариев из закладок браузера, закладки применяемые так принято называть букмарклетами). О подключении сценария посредством псевдопротокола необходимо помнить следующее: разные браузеры по разному на это реагируют (Firefox затирает текущий документ результатом

такого сценария (если он есть), а прочие браузеры просто отбрасывают результат такого сценария).
Сценарии, подключенные (какими- либо способами) к одному документу, исполняются в контексте одного глобального объекта, и соответственно имеют доступ к одному объекту документа. Каждому веб- документу, открытому в браузере, соответствует свой глобальный объект и объект документа. И не важно, как открыт в браузере веб- документ — как окно (вкладка) или фрейм. Важно лишь, что при подключении к документу сценария, этому документу ставится в соответствие глобальный объект и объект документа. Когда же сценарий запускается из закладки, то он исполняется в контексте глобального объекта именно текущего веб- документа.
При подключении сценариев необходимо помнить также следующее — чтобы сценарию было доступно объектное представление древа документа, оно должно быть уже построенным; синхронным сценариям может быть доступна только часть древа — по элемент `<head>` включительно (так как именно он и инкапсуллирует элементы сценариев).

Об интерпретации сценариев

Сценарии интерпретируются в следующем порядке:
загрузка веб- документа начинается; создается объект `Document` – это объект, инкапсуллирующий представление веб- документа в интерпретаторе; представление элементов веб- документа инкапсуллируется в объектах `Element`; если в веб- документе встречаются элементы `<script>` (загружаемые синхронно с документом), то их представления тоже добавляются, и эти сценарии незамедлительно выполняются, и только после этого загрузка документа продолжается; состояние готовности документа — `document.readyState` равно `loading`;
загрузка документа продолжается, если приэтом в веб- документе встречаются сценарии (загружаемые асинхронно с документом), то их представления тоже добавляются, но эти сценарии исполняются только после завершения загрузки документа; по завершении загрузки документа состояние готовности документа — `document.readyState` равно `interactive`;
после загрузки документа завершается загрузка его дополнительного содержимого (такого как изображения); после исполнения сценариев и завершения загрузки дополнительного содержимого состояние готовности документа — `document.readyState` равно `complete`; по факту этого состояния готовности в глобальном объекте возникает событие `load`; по факту этого события становятся доступны зарегистрированные обработчики событий;

Эти состояния готовности поддерживаются всеми браузерами, однако браузеры могут поддерживать и дополнительные, специфичные им состояния готовности.
Что касается исполнения сценариев:
сценарии синхронные поддерживались языком изначально — это самый старый способ подключения сценариев; предполагалось, что сценарии будут проделывать свою работу синхронно с загрузкой документа, и поэтому

предполагалось что они могут изменять документ при помощи соответствующих методов (документ, еще не загруженный до конца, изменять считается еще целесообразным); этим сценариям документ видим безусловно до их элемента `<script>` включительно;
сценарии асинхронные (собственно асинхронные) стали поддерживаться языком в последствии (необходимость в них возникла потому, что сценарии постепенно усложнялись, и их исполнение стало надолго задерживать загрузку документа); предполагалось, что достаточно сложные сценарии следует исполнять после загрузки документа, и поэтому предполагалось что они не могут изменять документ (уже загруженный документ изменять считается не целесообразным); этим сценариям документ видим безусловно до их элемента `<script>` включительно;
сценарии асинхронные (отложенные) в основном идентичны собственно асинхронным; важнейшее различие — отложенным сценариям документ видим безусловно полностью (действительно, для сценариев, исполняемых после загрузки документа, возможность видеть весь документ весьма целесообразна — и поэтому она была в последствии внедрена);

Асинхронные сценарии, различные в плане синхронности, внедрены в язык в разное время — во многом этим и обусловлены их различия.

О политике общего происхождения

Политика общего происхождения подразумевает следующее — сценарий может взаимодействовать только с документами, имеющими общее происхождение с документом, в который встраивается этот сценарий. При этом, происхождение самого сценария может быть любым.
Само понятие происхождения тоже требует комментариев. Происхождение в данном случае подразумевает следующее: протокол, по которому загружен документ (причем, разное значение, прописанное в этом поле, однозначно говорит о разных протоколах — `http` и `https` это разные протоколы); URL загруженного документа (конкретно, доменная составляющая URL); номер порта URL- адреса загруженного документа. Если хотя бы некоторые составляющие происхождения разные, то происхождения считаются разными.

Объект Window

`window.location`

используется для ссылки на объект `Location`;
`Location`, в свою очередь, служит для инкапсуллирования URL текущего документа;
ссылка `window.location` равна ссылке `document.location`; объект `Document` содержит также ссылку `document.URL`, она ссылается на строку с URL текущего документа; этому члену глобального объекта можно присвоить

как абсолютный адрес, так и относительный — и тогда он будет относительно текущего адреса; если этот член реинициализировать — то это приведет к загрузке документа по указанному адресу;
имеются и прочие члены этого объекта, в частности методы (в частности, для явной загрузки документа) — о них в справочнике;

window.history

используется для ссылки на объект `History`;
этот объект служит для хранения истории браузера; сценарии не имеют прямого доступа к нему (из соображений безопасности); но они имеют к нему косвенный доступ;
методы `history.back()`, `history.forward()` имеют действие одноименных кнопок браузера; метод `history.go()` используется для перемещения на адрес `+число` или `-число` в списке истории относительно текущего адреса; для работы таких методов языком поддерживается объект `history.lenght` – но, как и говорилось выше, этот объект недоступен сценарию (из соображений безопасности);

window.navigator

используется для ссылки на объект `Navigator`;
этот объект инкапсуллирует общие сведения о браузере; он имеет такое имя по историческим причинам (назван так в честь одноименного браузера от известной компании); `navigator.appName` – строка (название браузера, это специфика конкретного браузера), `navigator.appVersion` – строка (версия браузера, это специфика конкретного браузера), `navigator.userAgent` – строка (посылается в http- заголовке `USER-AGENT`), `navigator.platform` – строка (содержит описание платформы, это специфика конкреной платформы);
`navigator.online` – определяет есть ли соединение, `navigator.geolocation` – определяет геоположение, `navigator.javaEnabled` – определяет поддержку апплетов, `navigator.cookiesEnabled` – определяет разрешены ли куки;

window.screen

используется для ссылки на объект `Screen`;
этот объект инкапсуллирует общие сведения об экране компьютера;
`width` – ширина экрана (в пикселах), `height` – высота экрана (в пикселах), `availWidth` – доступная (за вычетом элементов ГПИ) ширина экрана (в пикселах), `availHeight` – доступная (за вычетом элементов ГПИ) высота экрана, `colorDepth` – глубина цвета (в битах на пиксел);

`window.document`

используется для ссылки на объект `Document`; этот объект инкапсуллирует объектное представление веб- документа; и, вложенность элементов документа безусловно отражается в их объектном представлении; причем, контент элемента тоже имеет объектное представление (эти объекты содержат строку их контента, и (по понятной причине) не могут инкапсуллировать объекты элементов документа); пробельные символы (даже употребленные сами по себе, в целях соблюдения code conventions) в контенте элемента тоже считаются его контентом; касательно пробельных символов стоит заметить также следующее: символы, следующие за `</body>` получат объектное представление в объекте (не вне объекта тела, а внутри него) — дабы не было иллюзии, что контент документа пребывает вне его тела, символы перед `<head>` не получают объектного представления вообще (это по историческим причинам); объекты — представления элементов (обычно) имеют одинаковые с ними имена (они пишутся строчными ввиду регистронезависимости `HTML`), однако конкретные элементы могут иметь другие имена; если же речь идет о документе, составленном не корректно (в плане тегов, образующих его структуру), то объектное представление такого документа все равно будет построено (причем, в объектном представлении документ будет иметь уже корректную структуру); корректность структуры документа определяется в соответствии со спецификацией DOM (что в отношении таблиц означает следующее — в объектном представлении таблицы всегда будет присутствовать `<tbody>`); элементы комментариев также получают объектное представление — в виде объектов комментариев; элемент `<!DOCTYPE>`, предваряющий древо `<HTML>` также получает объектное представление — в виде объекта элемента;

Объект Window. Таймеры

Это функции, позволяющие вызвать какую- либо функцию в заданное время.

`window.setTimeout()`

используется для запуска указанной функции в указанное время; принимает параметры — функцию (указанная функция), и число (число миллисекунд, спустя которое указанная функция будет вызвана); планирует запуск указанной функции, руководствуясь переданными параметрами; возвращает число (переданное число миллисекунд);

`window.clearTimeout()`

используется для отмены действия функции `setTimeout()`; принимает параметр — функцию, указанную функции `setTimeout()`; отменяет

действие функции `setTimeout()` для указанной ей функции; возвращает ничтоже;

`window.setInterval()`

используется для запуска указанной функции с указанным интервалом; принимает параметры — функцию (указанная функция), и число (число миллисекунд, указанная функция будет запускаться итеративно, с этим интервалом); планирует запуск указанной функции, руководствуясь переданными параметрами; возвращает число (переданное число миллисекунд);

`window.clearInterval()`

используется для отмены действия функции `setInterval()`; принимает параметр — функцию, указанную функции `setInterval()`; отменяет действие функции `setInterval()` для указанной ей функции; возвращает ничтоже;

Объект Window. Диалоги

Речь идет о диалоговых окнах. До закрытия диалогового окна функция (выведшая на экран окно) продолжает исполняться, и дальнейший код сценария не продолжит исполняться до закрытия этого окна.

`window.alert()`

используется для вывода диалогового окна (текст самого окна, кнопка «ok»); принимает параметры — строку (отображается в диалоговом окне); в результате взаимодействия с юзером просто закрывает окно; возвращает ничтоже;

`window.confirm()`

используется для вывода диалогового окна (текст самого окна, кнопки «ok» и «cancel»); принимает параметр — строку (отображается в диалоговом окне); в результате взаимодействия с юзером возвращает логическое значение (`true` — если «ok», `false` – если «cancel»); возвращает логическое значение;

`window.prompt()`

используется для вывода диалогового окна (поле ввода, текст самого окна, кнопки «ok» и «cancel»); принимает параметры — строку (отображается в самом диалоговом окне), и строку (она по умолчанию заполняет поле ввода диалогового окна); в результате взаимодействия с юзером возвращает либо

строку (пользовательский ввод) — если «ok», либо специальное значение `null` – если «cancel»; возвращает значение (`String` или `null`);

`window.showModalDialog()`

используется для вывода диалогового окна (в качестве контента окна используется веб- документ); принимает параметры — строку (с адресом веб-документа), значение (ссылочного типа, его предполагается использовать в сценарии, подключенном к этому документу — оно иниц- ет `window.dialogArguments`), и строку (трактуется как список пар (список имеет вид `Name:Value; …`), этот список служит для передачи настроек диалогового окна, и представляет собой браузерно- специфичную информацию); выводит (руководствуясь переданными параметрами) диалоговое окно в новом окне/ вкладке; возвращает ничтоже (но иниц- ет `window.returnValue`); устарел со времен FireFox56/Chrome43;

`window.onerror`

используется для ссылки на пользовательский обработчик прерываний; это обработчик прерываний наивысшего уровня их вложенности — если прерыванию удалось распространиться до глобального уровня, то этот обработчик будет последним (пользовательским) обработчиком, который может его перехватить; этот обработчик должен прописываться следующим образом — ссылке на него присваивается функциональное выражение, это функциональное выражение должно принимать параметры — строку (сообщение о произошедшей ошибке) и строку (адрес документа- сценария — источника ошибки) и строку (номер строки в документе- сценарии — источнике ошибки), аргументы должны передаваться именно так (и причем явно), исполнять прописанное программистом тело, и возвращать логическое значение (истина — при факте перехвата); это средство устарело — оно является унаследованным (в свое время в языке `JavaScript` просто не было конструкций обработки прерываний);

Элементы с атрибутом id

Если при интерпретации веб- документа обнаруживается элемент с атрибутом `id`, то глобальному объекту автоматически будет приписан член с именем — значением этого атрибута. Значением этого члена будет объект `HTMLElement` – объект из древа документа. Этот член будет не перечислимым.
Однако, если член с этим именем уже присутствует в глобальном объекте, то никакого такого автоматического приписывания членов не произойдет. Если же такое автоматическое объявление происходит, то в дальнейшем сценарий все же может объявить член с таким же именем (тогда автоматически объявленный член будет просто переопределен). Такое поведение интерпретатора поддерживается в настоящее время из соображений обратной совместимости. И, в настоящее время, эта возможность считается (по большому счету) устаревшей.

Такое же поведение интерпретатора наблюдается в следующем случае: если элементы веб- документа (<a>, <area>, <frame>, <frameset>, <iframe>, <img>, <form>, <object>, <embed>) прописаны с атрибутом name. Если же при этом ряд элементов будет иметь одно и то же значение атрибута name (или если атрибут name одного элемента равен атрибуту id другого), то будет автоматически объявлена одна переменная с этим именем. И она будет (индексируемым) объектом, инкапсуллирующим объекты этих элементов. При этом, необходимо помнить следующее — элементы <iframe> с атрибутом id или name, при обнаружении приводят к (автоматическому) объявлению другой переменной — объекта Window (инкапсуллирующего этот фрейм).

Работа с несколькими окнами/вкладками/фреймами

Взаимодействие сценариев и веб- документов ограничивается политикой общего происхождения. Таким образом, в общем случае, сценарию (подключенному к одному документу) не запрещается взаимодействовать с другими документами.
Чтобы с объектом окна можно было работать, он должен быть создан (действительно, нельзя работать с несуществующим объектом).
Каждое окно/ вкладка/ фрейм является объектом window. Этот объект, как и говорилось выше, является глобальным. Таким образом, разные окна/ вкладки/ фреймы — это разные глобальные объекты; и, соответственно, разные области видимости. Исключение составляют вложенные объекты window (они бывают вложенными в случае использования (обычных или плавающих) фреймов) – вложенные объекты window представляют собой единую область видимости (именно единую, а не инкасуллирующую и инкапсуллируемые). В этом отношении ко вложенным объектам приравниваются объекты- потомки (объект окна ситается потомко другого объекта окна, если он был им открыт).
Для ссылки из объекта окна на его родителя служит window.parent (window. … .parent – для соответствующего уровня иерархии предков). Для фреймов и окон также доступен член window.top – он указывает на верх иерархии фреймов.
Явно создать объект window из кода сценария можно методом window.open().

window.open()

используется для открытия окна; принимает параметры — строку (URL, этот параметр может отсутствовать — тогда будет открыто пустое окно), строку (имя окна, этот параметр может отсутствовать — тогда окно получит имя _blank, если же этот параметр присутствует и при этом указывает имя уже существующего окна — то оно и будет использоваться), строку (список параметров окна (имеет вид Name=Value, …), это браузерно- специфичная информация, этот параметр может отсутствовать), и логическое значение (определяет, должен ли адрес открываемого окна заменить последнюю запись в

истории (true) или же дописать историю (false)); открывает окно (руководствуясь переданными параметрами); возвращает объект window (это открытое окно);
обычно этот метод завершается самим браузером — по причине борьбы со всплывающими окнами (чтобы этот метод сработал успешно, он должен вызываться в ответ на действия пользователя);

Окна, открытые таким способом, получают особое значение члена window.opener – в данном случае это значение ссылки на объект окна, сценарий из которого открыл данное окно; иначе же это значение — специальное значение null.
Для закрытия окон используется метод window.close(). Если он вызывается из окна, которое подлежит закрытию, то он вызывается его собственный метод. Если же он вызывается из окна, открывшего его — то он должен быть вызван как метод этого объекта.
Этот метод, при вызове из своего или родительского окна просто делает свою работу; при вызове из какого- либо другого окна он потребут подтверждение на закрытие от пользователя.

```
window.close()
```

используется для закрытия окна; параметров не принимает; закрывает свое окно; возвращает ничтоже;
применительно к (каким- либо) фреймам этот метод просто не производит действий (считается, что фреймы непременно должны отображаться — пользоваться веб- документом, на котором закрыты некоторые фреймы, может быть проблематично);

Объект Document

Как и говорилось выше, этот объект предназначен для инкапсуллирования объектного представления документа.
Терминология, используемая касательно узлов документа (такие термины, как родительский, дочерний, …), стандартная, - та же терминология используется в томе HTML, CSS.
Прежде всего, целесообразно оговорить иерархию (в плане прототипно-ориентированного наследования) различных узлов объектного представления документа.
Объект Node представляет собой верх иерархии различных узлов объектного представления документа. От него наследует объект Document (причем, речь идет не об объекте, инкапсуллирующем документ, а о служебном объекте типа Document). От него, в свою очередь, наследует объект HTMLDocument (этот объект имеет именно этот тип (в случае HTML- документа), это и будет сам инкапсуллирующий объект, доступный по ссылке window.document). От Node так же наследует объект Element (служебный объект (служащий для хранения

общей для наследующих объектов информации)). От него, в свою очередь, наследует объект `HTMLElement` (этот объект имеет именно этот тип (в случае `HTML`- документа), это служебный объект (служащий для хранения общей для наследующих объектов информации)). От объекта `HTMLElement` наследуют объекты `HTMLHeadElement`, `HTMLBodyElement`, … - объекты конкретных элементов. От `Node` так же наследует объект `CharacterData` (служебный объект (служащий для хранения общей для наследующих объектов информации)). От него, в свою очередь, наследуют объекты `Text` (это объекты, инкапсуллирующие контент элементов документа), и объекты `Comment` (это объекты, инкапсуллирующие элементы комментариев). От `Node` так же наследует объект `Attr` (объект, инкапсуллирующий атрибуты элементов документа; в практике программирования не используется, так как для доступа к атрибутам элементов используются методы объектов- элементов).

Получение объеков- элементов

Для явного получения объектов- элементов используются соответствующие методы объекта `document`. Это следующие методы.

`document.getElementById()`

используется для получения объекта- элемента документа; принимает параметр — строку (содержит значение атрибута `id` элемента документа); возвращает объект- элемент своего документа, у которого указанный атрибут имеет указанное значение; возвращает объект (объект- элемент);

`document.getElementsByName()`

используется для получения объекта- элемента документа; принимает параметр — строку (содержит значение атрибута `name` элементов документа); возвращает объекты- элементы своего документа, у которых указанный атрибут имеет указанное значение; возвращает объекты- элементы (в составе объекта `NodeList` – это объект, подобный массивам, и доступный только для чтения); этот метод доступен только для `HTML`- документов;

`document.getElementsByTagName()`

используется для получения объекта- элемента документа; принимает параметр — строку (содержит имена самих элементов элементов документа); возвращает объекты- элементы своего документа, у которых само их имя имеет указанное значение; возвращает объекты- элементы (в составе объекта `NodeList` – это объект, подобный массивам, и доступный только для чтения);

так как имена элементов в коде HTML регистронезависимы, то и сравнение их со строкой (в параметрах) будет регистронезависимым;
есть возможность запросить все элементы, для этого в параметрах нужно передать строку с символом *;
этот метод доступен также из объектов Element, реализация в них имеет следующую специфику — будет возвращен массив объектов- элементов, являющихся потомками своего элемента данного метода;

```
document.getElementsByClassName()
```

используется для получения объекта- элемента документа; принимает параметр — строку (содержит значение атрибута class элементов документа); возвращает объекты- элементы своего документа, у которых атрибут class имеет указанное значение; возвращает объекты- элементы (в составе объекта NodeList – это объект, подобный массивам, и доступный только для чтения);
при использовании этого метода необходимо помнить, что если браузер находится в режиме совместимости, то сравнение со строкой (в параметрах) будет проистекать регистронезависимо;

```
document.querySelectorAll()
```

используется для получения объекта- элемента документа; принимает параметр — строку (содержит селектор (CSS) элементов документа); возвращает объекты-элементы своего документа, у которых селектор имеет указанное значение; возвращает объекты- элементы (в составе объекта NodeList – это объект, подобный массивам, и доступный только для чтения);
при использовании этого метода необходимо помнить, что объект, возвращаемый этим методом не является «живым»;
этот метод не работает с псевдоэлементами и псевдоклассами;
этот метод доступен также из объектов Element, реализация в них имеет следующую специфику — будет возвращен массив объектов- элементов, являющихся потомками своего элемента данного метода;

```
document.querySelectorAll()
```

используется для получения объекта- элемента документа; принимает параметр — строку (содержит селектор (CSS) элемента документа); возвращает объект-элемент своего документа, первый из тех, у которых селектор имеет указанное значение; возвращает объект- элемент (в составе объекта NodeList – это объект, подобный массивам, и доступный только для чтения);
при использовании этого метода необходимо помнить, что объект, возвращаемый этим методом не является «живым»;
если не найден ни один соответствующий элемент, то этот метод вернет не пустой объект NodeList, а специальное значение null;

этот метод не работает с псевдоэлементами и псевдоклассами;
этот метод доступен также из объектов `Element`, реализация в них имеет следующую специфику — будет возвращен массив объектов- элементов, являющихся потомками своего элемента данного метода;

Структура и навигация

Навигация по структуре документа осуществляется следующими средствами.

Объект Node

Конструктор объектов Node

Конструктора этих объектов не существует, так как эти объекты являются «статическими» объектами;

Методы объектов Node

Подробности — в справочнике.

Переменные объектов Node

`parentNode`

используется для ссылки на родительский узел данного узла;

`childNodes`

исползуется для ссылки на дочерние узлы (хранятся в объекте `NodeList`);

`firstChildren`

используется для ссылки на первый дочерний узел;

`lastChildren`

используется для ссылки на последний дочерний узел;

`previousSibling`

используется для ссылки на предыдущий сестринский узел;

`nextSibling`

используется для ссылки на следующий сестринский узел;

`nodeType`

используется для обозначения типа данного узла (это численное значение, для узла типа `Document` – `9`, `Element` – `1`, `Text` – `3`, `Comments` – `8`);

`nodeValue`

используется для хранения содержимого данного узла (причем, эта переменная доступна и для элементов комментариев), оно хранится в виде строки;

`nodeName`

используется для хранения имени элемента (своего узла);
причем, имя хранится независимо от того, как оно прописано в коде `HTML` (это сделано ради соблюдения конвенции), поэтому оно хранится в виде строки;

Объекты Element

Конструктор объектов Element

Конструктора этих объектов не существует, так как эти объекты являются «статическими» объектами;

Методы объектов Element

Подробности — в справочнике.

Переменные объектов Element

`children`

аналогично члену `childNodes` объекта `NodeList`, за тем исключением, что указывает только объекты `Element`;

`firstElementChild`

аналогичен члену `firstChild` объекта `NodeList`, за тем исключением, что указывает только объекты `Element`;

`lastElementChild`

аналогичен члену `lastChild` объекта `NodeList`, за тем исключением, что указывает только объекты `Element`;

`nextElementSibling`

аналогичен члену `nextSibling` объекта `NodeList`, за тем исключением, что указывает только объекты `Element`;

`previousElementSibling`

аналогичен члену `previousSibling` объекта `NodeList`, за тем исключением, что указывает только объекты `Element`;

`childElementCount`

содержит количество дочерних элементов (это численное значение);

Атрибуты элементов

Объекты элементов имеют члены, представляющие собой атрибуты этих элементов. Обратиться к ним можно обычной для этого записью (записью `ObjectName.VarName`). Имена атрибутов элементов документа, ввиду регистронезависимости кода `HTML`, обычно прописываются в коде сценария во вполне предсказуемой нотации (имена вида `VARNAME` прописываются в виде `varname`, имена вида `VARNAME-VARNAME` – как `varnameVarname`, именно так). Значениями атрибутов обычно оказываются значения соответствующих типов (то есть, их типы соответствуют форме записи этих значений; но некоторые атрибуты имеют специфические типы значений).
К атрибутам элементов можно обратиться также следующим образом — через член `atributes` объекта- элемента. Этот член содержит ссылку на объект, подобный массиву; этот объект используется для ссылки на объекты `Attr` – они, в свою очередь, представляют собой атрибуты элемента (они были рассмотрены выше).
В языке `JavaScript` есть ряд специфических методов для работы с атрибутами элементов документа. Это следующие методы объектов элементов.

`getAttribute()`

используется для получения значения указанного атрибута; принимает параметр — строку (содержит имя целевого атрибута в языке `HTML`); возвращает значение указанного атрибута (в строковом представлении); возвращает строку;

`setAttribute()`

используется для установки значения указанного атрибута; принимает параметры — строку (содержит имя целевого атрибута в языке `HTML`), и строку (содержит значение целевого атрибута); устанавливает значение атрибута (руководствуясь переданными параметрами); возвращает ничтоже;

`hasAttribute()`

используется для проверки наличия указанного атрибута; принимает параметр — строку (содержит имя целевого атрибута в языке `HTML`); возвращает результат проверки (логическое значение); возвращает логическое значение;

`removeAttribute()`

используется для удаления указанного атрибута; принимает параметр — строку (содержит имя целевого атрибута в языке `HTML`); удаляет атрибут (руководствуясь переданными параметрами); возвращает ничтоже;

Содержимое элементов

Для доступа к содержимому элемента служат следующие средства.
Член `innerHTML` объекта- элемента содержит контент своего элемента; он доступен для чтения и записи, при доступе по записи ему присваивается то, что должно быть содержимым элемента. Член `outerHTML` объекта- элемента содержит полную запись своего элемента, включая запись тегов; он доступен для чтения и записи, при доступе по записи ему присваивается то, что должно быть записью элемента.
Член `textContent` объекта- элемента содержит контент своего элемента за исключением вложенных элементов; особенности работы с этим членом следует уточнять для конкретного браузера. Член `nodeValue` объекта- элемента содержит контент своего элемента за исключением вложенных элементов; он доступен для чтения и записи.

Изменение документа

Имеются следующие средства для изменения документа.

`document.createElement()`

используется для создания объекта- элемента; принимает параметр — строку (имя элемента, регистронезависимо); создает объект- элемент; возвращает объект- элемент;

`document.createTextNode`

используется для создания текстового узла; принимает параметр — строку (контент текстового узла); создает текстовый узел; возвращает текстовый узел;

`document.createComment`

используется для создания комментария; принимает параметр — строку (контент комментария); создает узел- комментарий; возвращает узел- комментарий;

`Element.cloneNode()`

используется для клонирования своего объекта- элемента; принимает параметр — логическое значение (`true` – клонировать вложенные узлы, `false` – не клонировать вложенные узлы); выполняет клонирование своего объекта-элемента (руководствуясь переданными параметрами); возвращает объект (клон своего объекта);
узел- клон не является частью документа, и не имеет узла родителя — пока не будет добавлен в узел, являющийся таковым (вставка осуществляется соответствующим методом);

`Document.importNode()`

используется для импортирования узла документа; принимает параметры — объект (импортируемый узел), и логическое значение (`true` – клонировать вложенные узлы, `false` – не клонировать вложенные узлы); импортирует узел документа, и для этого клонирует указанный узел (руководствуясь переданными параметрами); возвращает объект (импортируемый узел);
импортированный узел не является частью документа, и не имеет узла родителя — пока не будет добавлен в узел, являющийся таковым (вставка осуществляется соответствующим методом);

`Document.adoptNode()`

используется для вырезания узла документа; принимает параметр — объект (вырезаемый узел); выполняет вырезание указанного элемента из документа; возвращает объект (вырезанный узел);
вырезанный узел не является частью документа, и не имеет узла родителя — пока не будет добавлен в узел, являющийся таковым (вставка осуществляется соответствующим методом);

`Element.appendChild()`

используется для добавления узла в документ; принимает параметр — объект (добавляемый узел); добавляет указанный узел в свой узел в качестве дочернего (он будет добавлен в конец списка дочерних узлов); возвращает узел (добавляемый узел);

`Element.insertBefore()`

используется для вставки элемента; принимает параметры — объект (вставляемый элемент), и объект (элемент, относительно которого произойдет вставка); производит вставку элемента (руководствуясь переданными параметрами); возвращает объект (вставленный элемент);

`Element.replaceChild()`

используется для замещения элемента; принимает параметры — объект (замещающий элемент), и объект (замещаемый дочерний элемент); замещает свой дочерний элемент (руководствуясь переданными параметрами); возвращает объект (замещенный элемент);

`Element.removeChild()`

используется для удаления дочернего элемента; принимает параметр — объект (удаляемый дочерний элемент); удаляет указанный дочерний элемент; возвращает объект (удаленный элемент);

Объекты DocumentFragment

Объекты `DocumentFragment` служат пользовательскими контейнерами узлов документа. Такие контейнеры создаются следующей записью:

```
var VarName = document.createDocumentFragment();
```

созданный фрагмент является самостоятельным объектом (он не имеет узла-предка).

Эти объекты позволяют манипулировать их контентом как единым объектом. Методы `appendChild()`, `replaceChild()`, `insertBefore()` могут принимать эти объекты в аргументах, и тогда в качестве указанного элемента будет выступать контент объекта `DocumentFragment` (в результате этот контент будет необернутым в объект).

О формах

Элементы форм могут быть явно выбраны рассмотренными выше способами. Также они могут быть выбраны и следующим специфическим способом:

```
document.forms.Name;
//Name – значение атрибута name целевого элемента
```

также элементы форм могут быть выбраны индексированием (`document.forms[ … ]`), индекс соответствует порядку элемента в коде документа. Индексирование является альтернативным способом, употребляемым при работе с такими элементами, как checkbox forms, radio forms (так как эти элементы, идущие подряд, наделяются одним и тем же именем).
Член `elements` элемента формы ссылается на объект подобный массиву, инкапсуллирующий элементы внутри формы. Метод `submit()` элемента формы используется для отправки данных своей формы (из кода сценария), метод `reset()` элемента формы используется для сброса данных своей формы (из кода сценария).

Некоторые члены, общие для элементов форм

`name`

атрибут `name`;

`value`

это данные, которые отправляются из формы;

`type`

атрибут `type`;

`form`

ссылка на инкапсуллирующий элемент формы;

Всем формам, помимо прочих событий, доступны события `onsubmit` (это событие наступает при отправке данных формы) и `onreset` (наступает при сбросе данных формы). Обработчик события `onsubmit` вызывается перед отправкой данных формы; он вызывается только при отправке данных действиями пользователя (и не вызывается при отправке из сценария). То же касается и обработчика события `onreset`.

Некоторые члены объекта Document

`location`

дублирует `window.location`;

`lastModified`

строка (дата последнего изменения документа);

`referrer`

содержит адрес документа, по ссылке из которого был открыт текущий документ;

`title`

контент одноименного элемента документа;

`URL`

аналогично `location`, но доступно только для чтения;

`cookie`

подробнее- в справочнике;

`domain`

подробнее- в справочнике;

О событиях

Понятие события было прокомментировано выше.
О том или ином событии принято говорить также как о событии того- или иного типа (о событии `on...` принято говорить также как о событии типа `on...`). Подробнее об этих терминах — в спецификации (что касается прочих источников — то они могут использовать разную терминологию).
Возникновение события приводит к выбросу объекта соответствующего события; этот объект неявно передается обработчику этого события. Это делает события похожими на прерывания. Есть еще и сходство в распространении — подобно прерываниям, события могут распространяться (если у элемента наступило

событие, обработчик которого не зарегистрирован — то обработчик этого события ищется в инкапсуллирующих элементах).

Различные типы событий

Различные виды событий можно подразделить на следующие.
События ввода (аппаратно- зависимые) — наступают в результате действий пользователя (это события с точки зрения действий с устройствами ввода). События ввода (аппаратно- независимые) — наступают в результате действий пользователя (это события с точки зрения сути действий). События пользовательского интерфейса — наступают в результате действий пользователя (это такие события как focus, submit). События изменения состояния — наступают в результате наступления очередного этапа загрузки документа, или же какого- либо события (в процессе работы браузера). И прочие события.
Однако, эта (официальная) классификация мало удобна. Поэтому, события будут рассмотрены в следующей классификации.
Далее приведены различные события; они приведены по своим именам, а не именам обработчиков.

События форм

blur

это событие доступно элементам форм; оно наступает при прекращении фокуса на элементе формы;

focus

это событие доступно элементам форм; оно наступает при возникновении фокуса на элементе формы;

click

это событие доступно кнопкам, флажкам, и визуально сходным элементам; оно наступает при клике по ним;

change

это событие доступно элементам форм с полями для пользовательского ввода; оно наступает по завершении ввода;

`submit`

это событие доступно элементу формы; оно наступает при отправке данных формы (в результате действий пользователя);

`reset`

это событие доступно элементу формы; оно наступает при сбросе данных формы (в результате действий пользователя);

`События окна`

`load`

наступает по завершении загрузки документа (включая изображения);

`unload`

наступает по закрытию окна;

`beforeunload`

наступает предварительно событию `unload`, и позволяет его превентировать (запрашивая подтверждения пользователя);

`onerror`

наступает при распространении ошибки/ события до глобального уровня (при отсутствии специально предназначенного обработчика); устарело, и практически выведено из употребления;

`error`

наступает при возникновении ошибки в ходе загрузки;

`abort`

возникает при отмене загрузки документа пользователем;

`resize`

наступает при масштабировании окна браузера;

`scroll`

наступает при прокрутке контента окна браузера;

`focus`

наступает при фокусе на окне браузера;

`blur`

наступает при прекращении фокуса на окне браузера;

`События мыши`

`mousemove`

наступает при перемешении мыши;

`mouseover`

наступает при наведении указателя мыши;

`mouseuot`

наступает при снятии наведения указателя мыши;

`mousedown`

наступает при нажатии клавиши мыши;

`mouseup`

наступает при отпускании клавиши мыши;

`click`

наступает при клике;

`dbclick`

наступает при двойном клике;

`mousewheel`

наступает при прокручивании колеса мыши;

`События клавиатуры`

`keydown`

наступает при нажатии клавиши;

`keyup`

наступает при отпускании клавиши;

`keypress`

наступает при нажатии клавиши еще до ее отпускания (в случае клавиш, соответствующих печатаемым символам);

Регистрация обработчиков событий

Регистрация обработчиков событий (подключение событий) возможна несколькими способами.
Приписыванием соответствующего члена целевому объекту:
```
ObjectName.EventName = function() { … };
```

Приписыванием соответствующего события целевому элементу:
```
<ElName EventName=" … ">
```

если же событие требуется приписать таким способом всему окну, то оно приписывается элементу тела.
Методом объектов- получателей событий `addEventListener()`:

`addEventListener()`

используется для регистрации обработиков событий; принимает параметры — строку (содержит тип события, именно это значение), функцию (вызывается при возникновении этого типа события), логическое значение (считать ли обработчик перехватывающим (`true` – считать, `false` – нет)); регистрирует обработчик события (руководствуясь переданными параметрами); возвращает ничтоже;
этот метод позволяет зарегистрировать более одной переданной ему во втором аргументе функции (это делается секвенцией вызовов этого метода, и тогда все

эти функции будут вызваны в порядке их регистрации (при наступлении события указанного типа); секвенцией вызовов этого метода каждую функцию (2й арг- т) можно зарегистрировать только один раз — все последующие попытки повторной регистрации просто отбрасываются);
есть и метод противоположного действия — `removeEventListener()`, он принимает те же аргументы, и (руководствуясь ими) удаляет ранее зарегистрированный обработчик;

Вызов обработчиков событий

Обработчик события вызывается при наступлении события, и в контексте объекта- элемента, которому он приписан.
Обработчики событий имеют доступ к той области видимости, в которой они определены; если же обработчик события подключается в коде `HTML`, то он считается определенным в глобальной области видимости.
Обработчики событий могут возвращать значения. Эти значения возвращаются в целях их трактовки браузером (тем или иным способом). Эти значения — специфика конкретного браузера и конкретного события. Возвращаемые значения учитываются браузером только если обработчик был приписан как член объекта- получателя. Иначе же они просто отбрасываются; чтобы в таком случае иметь возможность передать (учитываемое) значение, это делается косвенно — установкой значения члена `returnValue` объекта- события.

HTTP

О протоколе HTTP

Протокол HTTP – протокол прикладного уровня стека протоколов TCP/IP. Как следует из названия (HyperText Transfer Protocol), он используется, главным образом, как протокол взаимодействия браузера и веб- сервера.
Protocol Data Unit этого протокола — это сообщение. Сообщение от браузера к серверу - «запрос», сообщение от сервера браузеру - «ответ».

Формат сообщения

Сообщение имеет следующий формат.

`Starting line`

определяет тип сообщения; стартовые строки запросов и ответов имеют разный формат; стартовая строка запроса имеет следующий формат — Method (метод запроса), URI (путь к запрашиваемому ресурсу), HTTP/Version (версия протокола HTTP); стартовая строка ответа имеет следующий формат — HTTP/Version

(версия протокола HTTP), Statuse Code (код состояния), Reason Phrase (текст комментария);

`Headers`

различные заголовки сообщения;

`Message body`

тело сообщения;

Методы запросов

Методы запроса могут быть следующими.

`OPTIONS`

используется для определения возможностей сервера;

`GET`

используется для запроса ресерса;

`HEAD`

используется для получения метаданных ресурса;

`POST`

используется для передачи данных на ресурс;

`PUT`

используется для передачи данных на ресурс;

`PATCH`

используется для передачи данных на ресурс (применительно к целевому фрагменту ресурса);

DELETE

используется для удаления целевого ресурса;

TRACE

используется для трассировки запросов;

CONNECT

используется для создания туннеля;

Различные серверы и клиенты могут поддерживать различные наборы методов, выше были рассмотрены лишь типовые.

Коды состояния

Код состояния говорит о результате исполнения запроса. Он представляет собой трехзначное число. Коды состояния подразделяются на классы (по первому разряду этих чисел). Пользовательский агент может и не знать всех кодов состояний, но должен знать, как трактовать коды в соответствии с их классами. Поддерживаются следующие коды состояний по классам.

1XX

коды класса Информационный (Informational); используются для передачи информации о процессе обмена данными;

2XX

коды класса Успех (Success); используются для информирования об успешном исполнении запроса;

3XX

коды класса Перенаправление (Redirection); используются для информирования о необходимости перенаправления;

4XX

коды класса Ошибка клиента (Client Error); используются для информировании об ошибке в запросе клиента;

`5XX`

коды класса Ошибка сервера (Server Error); используются для информирования об ошибке сервера;

Заголовки HTTP

Заголовки протокола HTTP (какие- бы то ни было) содержат данные в формате `Name:Value`.
Все заголовки подразделяются ледующим образом.

`General Headers`

общие заголовки, могут включаться в любое сообщение;

`Request Headers`

заголовки запроса, включаются в сообщение запроса;

`Response Headers`

заголовки ответа, включаются в сообщение ответа;

`Entity Headers`

заголовки сущности, включаются в сущность сообщения;

Прочие сведения о заголовках (для лиц, не участвующих в разработке сетевых протоколов) должны быть безынтересны.

Тело сообщения

Тело сообщения используется для передачи полезной нагрузки сообщения (поэтому, если вся полезная нагрузка — в заголовках, то тело отсутствует).
Тело сообщения не должны содержать ответы на запрос HEAD, ответы скодами состояния 1XX, 204 (No Content), 304 (Not Modified). Все прочие ответы содержат тело (хотя бы пустое).

Об использовании протокола HTTP

Обычно использование протокола `HTTP` происходит не явно, но в языке `JavaScript` имеются средства для явного его использования.

Объект XMLHttpRequest

Этот объект используется для явного использования протокола HTTP. Использование этих объектов не обязывает работать с файлами XML (эти объекты носят такое название сугубо по историческим причинам).
Для того, чтобы использовать такой объект, он должен быть создан:
```
var VarName = new XMLHttpRequest();
```

экземпляр этого объекта инкапсуллирует пару значений запрос/ ответ, и содержит переменные и методы для работы с этими значениями.
Эти объекты могут быть использованы многократно; но прервет исполнение текущего (с точки зрения использования этого объекта) запроса.
Для дальнейшего изучения материала следует повторить тему «О протоколе HTTP».
Объект `XMLHttpRequest`, по сути, есть средство по работе с запросами протокола HTTP. Однако, объект `XMLHttpRequest` есть средство, несколько абстрагирующееся над протоколом HTTP; таким образом, при использовании объекта `XMLHttpRequest` нет необходимости заботиться о ряде низкоуровневых вопросов (таких как кэширование, переадресация, куки, и пр.).
Объекты `XMLHttpRequest` работают с протоколом HTTP (и HTTPS), прочие протоколы (и псевдопротоколы) не используются (в частности, не используется псевдопротокол file://).
Для дальнейшей работы с созданным объектом `XMLHttpRequest` необходимо определить целевые параметры запроса. Для этого используется метод `open()` объекта `XMLHttpRequest`:
```
ObjectName.open( … );
```

метод `open()` используется для определения параметров запроса; принимает параметры — строку (метод запроса, регистронезависима), строку (URL, к которому планируется запрос); устанавливает данные поля запроса (руководствуясь переданными параметрами); возвращает ничтоже;
указывая метод запроса (из соображений безопасности) запрещено указывать следующие методы: HTTP CONNECT, TRACE, TRACK;
указывая URL (к которому планируется запрос) принято использовать относительные адреса;

для дальнейшего использования объекта `XMLHttpRequest` следует определить поле заголовков запроса. Для этого используется метод `setRequestHeader()` объекта `XMLHttpRequest`:
```
ObjectName.setRequestHeader( … );
```

метод `setRequestHeader()` используется для установки поля заголовков запроса; принимает параметры — строку (имя заголовка запроса), строку (значение заголовка запроса), строку (, этот аргумент опционален), строку (имя пользователя, этот аргумент опционален (используется когда для доступа к ресурсу требуется авторизация)), и строку (пароль, этот аргумент опционален

(используется когда для доступа к ресурсу требуется авторизация)); устанавливает поле заголовков запроса (руководствуясь переданными параметрами); возвращает ничтоже;
если вызвать этот метод несколько раз для одного и того же заголовка, этот заголовок будет не реинициализирован, а продублирован в поле заголовков запроса (либо этот заголовок будет инициализирован секвенцией указанных в вызовах метода значений);
допускается устанавливать только те заголовки, что важны пользователю (значимые сугубо для протокола заголовки нельзя устанавливать, такими считаются заголовки Content-Lenght, Date, Referer, User-Agent, и также Accept-Charset, Accept-Encoding, Connection, Cookie, Cookie2, Content-Transfer-Encoding, Expect, Host, Keep-Alive, TE, Trailer, Transfer-Encoding, Upgrade, Via);

для дальнейшего использования объекта `XMLHttpRequest` следует определить тело запроса (если оно фигурирует в запросе), и отправить запрос. Эти действия выполняются методом `send()` объекта `XMLHttpRequest`:

```
ObjectName.send( … );
```

если тело отсутствует, то в параметрах следует указать специальное значение `null`, либо ничего не указывать. По вызову этого метода запрос будет отправлен (с указанным в параметрах телом). Этот метод завершается по отправке запроса, и поступления ответа он не ждет.
Использование объекта `XMLHttpRequest` должно производиться в указанном порядке вышеописнных действий, и нарушать его нельзя.
Ответ на запрос содержит характерные поля, и для работы с ними используются следующие члены объекта `XMLHttpRequest`:

`status`

содержит код состояния;

`statusText`

содержит текст комментария;

`responseText`

содержит тело ответа;

`responseXML`

содержит тело ответа (в виде объекта `Document`);

`readyState`

содержит значение, определяющее код состояния (самого процесса исполнения запроса);
коды состояния:

0	UNSENT	метод `open()` не был вызван
1	OPENED	метод `open()` был вызван
2	HEADERS_RECEIVED	заголовки получены
3	LOADING	загрузка тела ответа
4	DONE	загружено тело ответа

браузеры (обычно) не вобуждают это событие, если код состояния 0 или 1;
когда это значение меняется, наступает событие `readystatechange`;

`getResponseHeader()`

используется для получения заголовков ответа;

`getAllResponseHeader()`

используется для получения заголовков ответа;

чтобы определить факт получения ответа необходимо обработать событие `readystatechange` (или `progress`) объекта `XMLHttpRequest`.

Cookies

Файлы cookie используются для хранения данных, значимых для сервера.
Название «cookies» происходит от жаргонизма «magic cookie» (означает сохраненные где- либо в ПО настройки).
Файлы cookie приходят клиенту при загрузке определенной страницы, и возвращаются серверу при обмене данными с ним; обмен этими данными (данными cookie) между браузером и сервером осуществляется в соответствующих заголовках HTTP (cookie отправляются клиенту когда их отправит сервер (и если настройки клиента это позволяют), и возвращаются серверу в каждом запросе к нему).
Сами файлы cookie представляют собой следущие данные: имя, значение, и атрибуты (используются в интересах дальнейшей эксплуатации файла); одна пара имя- значение (и атрибуты) хранятся в одном файле. Браузер должен поддерживать хранение не менее 300 таких файлов (и не менее 20 файлов с одного сервера), при их длине не менее 4Kb.
Прежде чем начат работу с файлами cookie целесообразно проверить, разрешил ли пользователь их сохранять — это делается проверкой (нестандартизованного

члена) navigator.cookieEnabled (содержит логическое значение; true – да, false – нет).
Срок хранения определяет собственно срок хранения. По уолчанию, все cookie – временные (удаляются при закрытии браузера). Срок хранения задается атрибутом max-age (в секундах).
Область видимости cookie определяет их доступность для сценариев. По умолчанию cookie доступны для сценариев страницы (при открытии которой они поступили) и страниц (хранящихся в том же каталоге, и его подкаталогах). Атрибут path задает URL (лежащие по которому страницы будут иметь доступ к данному файлу cookie), этот URL задается относительно домена сайта. Атрибут domain позволяет задать доступность cookie из поддоменов сервера, с которого поступил данный cookie (запись вида «.L2Domain.L1Domain» разрешит доступ из любых поддоменов L2Domain).
Безопасность cookie определяет возможность их передачи по незащищенному протоколу. Атрибут secure позволяет задать безопасность (это логическое значение).
Для доступа к cookie служит document.cookie.
Из сценария можно создать cookie; тогда он будет считаться поступившим при открытии соответствующей страницы. Это делается следующим путем:

```
document.cookie = "VarName=Value";
```

обращение по записи просто приводит к инициализации (или реинициализации) VarName; в этой записи можно также задать различные атрибуты (тогда строка примет вид «VarName0=Value0; AttrName1=Value1; … »).
Созданные cookie можно реинициализировать, для этого нужно применить такуюже запись (и в ней реинициализировать VarName).
Также можно реинициализировать значение атрибута max-age (это делается при реинициализации cookie).
Удалить cookie можно записью реинициализации, в которой VarName реинициализируется, и атрибут max-age инициализируется значением 0.
Когда к document.cookie обращаются по чтению, то возвращаяется строка, которая содержит список всех VarName, и имеет вид «VarName0=Value0; VarName1=Value1; … ». Атрибуты в этой строке не фигурируют.

Управление стилями (CSS)

Подробности — в справочнике.

Прочие средства

Подробности — в справочнике.