08 - Клиентский JavaScript, объект window, Web Storage

1. Выполнение JavaScript-программ

Вообще говоря, не существует формального определения программы на клиентском языке JavaScript. Можно лишь сказать, что программой является весь программный код на языке JavaScript, присутствующий в веб-странице (встроенные сценарии, обработчики событий в разметке HTML и URL-адреса javascript:), а также внешние сценарии JavaScript, на которые ссылаются атрибуты src тегов <script>. Все эти отдельные фрагменты программного кода совместно используют один и тот же **глобальный объект Window**. Это означает, что все они видят один и тот же объект Document и совместно используют один и тот же набор глобальных функций и переменных: если сценарий определяет новую глобальную переменную или функцию, эта переменная или функция будет доступна любому программному коду на языке JavaScript, который будет выполняться после этого сценария.

Программы на языке JavaScript выполняются в два этапа. **На первом этапе** производится загрузка содержимого документа и запускается программный код в элементах <script> (и встроенные сценарии, и внешние). Обычно сценарии выполняются в порядке их следования в документе. Внутри каждого сценария программный код выполняется последовательно, от начала до конца, с учетом условных инструкций, циклов и других инструкций управления потоком выполнения.

После загрузки документа и выполнения всех сценариев начинается **второй этап выполнения JavaScript-программы**, асинхронный и управляемый событиями. На протяжении этого этапа, управляемого событиями, веб-броузер вызывает функции обработчиков в ответ на события, возникающие асинхронно. Обычно обработчики событий вызываются в ответ на действия пользователя (щелчок мышью, нажатие клавиши и т.д.), но могут также вызываться в ответ на сетевые взаимодействия, по истечении установленного промежутка времени или при возникновении ошибочных ситуаций в JavaScript-коде.

Одно из первых событий, возникающих на управляемом событиями этапе выполнения, является событие load, которое сообщает, что документ полностью загружен и готов к работе. JavaScript-программы нередко используют это событие как механизм запуска. На практике часто можно увидеть программы, сценарии которых определяют функции, но не выполняют никаких действий, кроме определения обработчика события **onload**, вызываемого по событию load и запускающего управляемый событиями этап выполнения. Именно обработчик события onload выполняет операции с документом и реализует все, что должна делать программа.

```
var script = document.createElement('script');
script.src = "https://example.com/404.js"
document.body.appendChild(script);

script.onerror = function() {
      alert( "О ш и б к а : " + this.src );
};
```

Этап загрузки JavaScript-программы протекает относительно быстро, обычно он длится не более одной-двух секунд. Управляемый событиями этап выполнения, наступающий сразу после загрузки документа, длится на протяжении всего времени, пока

документ отображается веб-браузером. Поскольку этот этап является асинхронным и управляемым событиями, он может состоять из длительных периодов отсутствия активности, когда не выполняется никакой программный код JavaScript, перемежающихся всплесками активности, вызванной действиями пользователя или событиями, связанными с сетевыми взаимодействиями.

```
window.onload = function() { ... };

document.getElementById("button1").onclick = function() { ... };
```

2. Однопоточная модель выполнения

Обе разновидности языка, базовый JavaScript и клиентский JavaScript, поддерживают однопоточную модель выполнения. Сценарии и обработчики событий выполняются последовательно, не конкурируя друг с другом. Выполнение в единственном потоке существенно упрощает разработку сценариев: можно писать программный код, пребывая в полной уверенности, что два обработчика событий никогда не запустятся одновременно. Можно манипулировать содержимым документа, точно зная, что никакой другой поток выполнения не попытается изменить его в то же самое время, и вам никогда не придется беспокоиться о блокировках, взаимоблокировках или о состояниях гонки за ресурсами при разработке своих программ.

Выполнение в единственном потоке означает, что веб-броузер должен прекратить откликаться на действия пользователя на время выполнения сценария или обработчика события. Это накладывает определенные требования: сценарии и обработчики событий в JavaScript не должны исполняться слишком долго. Если сценарий производит объемные и интенсивные вычисления, это вызовет задержку во время загрузки документа, и пользователь не увидит его содержимое, пока сценарий не закончит свою работу. Если продолжительные по времени операции выполняются в обработчике события, броузер может оказаться не способным откликаться на действия пользователя, заставляя его думать, что программа «зависла».

3. Сценарии JavaScript в веб-приложениях

Веб-приложения применяют все возможности JavaScript и DHTML, которые используются в веб-документах, но помимо управления содержимым, его представлением и поведением они также используют преимущества других фундаментальных механизмов, предоставляемых веб-браузерами.

Чтобы понять суть веб-приложений, важно осознать, что веб-браузеры развивались не только как инструменты для отображения документов и давно уже трансформировались в некоторое подобие простых операционных систем. Сравните: традиционные операционные системы позволяют создавать ярлыки (представляющие файлы и приложения) на рабочем столе и в папках. Веб-браузеры позволяют создавать закладки (представляющие документы и веб-приложения) на панели инструментов и в папках. Операционные системы выполняют множество приложений в отдельных окнах; веб-браузеры отображают множество документов (или приложений) в отдельных вкладках. Операционные системы определяют низкоуровневые API для организации сетевых взаимодействий, рисования графики и сохранения файлов. Веб-браузеры определяют низкоуровневые API для организации сетевых взаимодействий, сохранения данных и рисования графики. Представляя веб-браузеры как упрощенные операционные системы, веб-приложения можно определить как веб-страницы, в которых используется

программный код на языке JavaScript для доступа к расширенным механизмам (таким как сетевые взаимодействия, рисование графики и сохранение данных) браузеров. Самым известным из этих механизмов является объект XMLHttpRequest, который обеспечивает сетевые взаимодействия посредством управляемых HTTP-запросов. Веб-приложения используют этот механизм для получения новой информации с сервера без полной перезагрузки страницы. Спецификация HTML5 и ряд связанных с ней спецификаций определяют ряд других важных прикладных интерфейсов для веб-приложений. В их число входят прикладные интерфейсы для сохранения данных и рисования графики, а также множество других возможностей, таких как геопозиционирование (geolocation), управление журналами посещений и фоновые потоки выполнения.

Различия веб-приложений и веб-документов. Безусловно, сценарии на языке JavaScript занимают в веб-приложениях более важное положение, чем в веб-документах. Сценарии на JavaScript расширяют возможности веб-документов, но при правильном оформлении документы остаются полностью доступными даже при отключенной поддержке JavaScript. Веб-приложения по определению являются программами на языке JavaScript, использующими механизмы, предоставляемые веб-браузерами, и не будут работать при отключенной поддержке JavaScript.

4. Объект Window

Объект Window является глобальным объектом для клиентских JavaScript-программ. В этой главе будут рассмотрены свойства и методы объекта Window. Эти свойства определяют множество различных API, из которых лишь немногие имеют отношение к окну браузера, в честь которого этот объект получил имя Window.

Таймеры. Функции **setTimeout()** и **setInterval()** позволяют зарегистрировать функцию, которая будет вызываться один или более раз через определенные интервалы времени. Это очень важные функции для клиентского JavaScript, и поэтому они были определены как методы объекта Window, несмотря на то что являются универсальными функциями, не выполняющими никаких действий с окном.

Метод setTimeout() объекта Window планирует запуск функции через определенное число миллисекунд.

Метод setInterval() похож на setTimeout(), за исключением того, что он автоматически заново планирует повторное выполнение через указанное количество миллисекунд:

```
setInterval(updateClock, 60000); // Вызывать updateClock()
через каждые 60 сек.
```

Подобно setTimeout(), и setInterval() возвращает значение, которое может быть передано методу **clearInterval()**, чтобы отменить запланированный запуск функции.

Адрес документа и навигация по нему. Свойство **location** объекта Window ссылается на объект Location, представляющий текущий URL-адрес документа, отображаемого в окне и определяющий методы, инициирующие загрузку нового документа в окно.

Свойство href объекта Location – это строка, содержащая полный текст URL-адреса. Другие свойства этого объекта, такие как protocol, host, hostname, port, pathname, search и hash, определяют отдельные части URL-адреса. Они известны как

свойства «декомпозиции URL». **Свойства hash** и **search** объекта Location представляют особый интерес. Свойство hash возвращает «идентификатор фрагмента» из адреса URL, если он имеется: символ решетки (#) со следующим за ним идентификатором. Свойство search содержит часть URL-адреса, следующую за вопросительным знаком, если таковая имеется, включая сам знак вопроса. Обычно эта часть URL-адреса является строкой запроса.

Загрузка нового документа. Метод **assign()** объекта Location заставляет окно загрузить и отобразить документ по указанному URL-адресу. Метод **replace()** выполняет похожую операцию, но перед открытием нового документа он удаляет текущий документ из списка посещавших страниц. Когда сценарию просто требуется загрузить новый документ, часто предпочтительнее использовать метод replace(), а не assign(). В противном случае кнопка Back (Назад) броузера вернет оригинальный документ и тот же самый сценарий снова загрузит новый документ. Примечательно, что строка URL-адреса в этом примере, переданная методу replace(), представляет относительный адрес. Относительные URL-адреса интерпретируются относительно страницы, в которой они появляются, точно так же, как если бы они использовались в гиперссылке.

Кроме методов assign() и replace() объект Location определяет также метод **reload()**, который заставляет броузер перезагрузить документ.

Однако более традиционный способ заставить броузер перейти к новой странице заключается в том, чтобы просто присвоить новый URL-адрес свойству location:

```
location = "http://www.oreilly.com"; // Перейти, чтобы
купить несколько книг!
```

Свойству location можно также присваивать относительные URL-адреса. Они разрешаются относительно текущего URL:

```
location = "page2.html"; // Загрузить следующую
страницу
```

История посещений. Свойство history объекта Window ссылается на объект History данного окна. Объект History хранит историю просмотра страниц в окне в виде списка документов и сведений о них. Свойство length объекта History позволяет узнать количество элементов в списке, но по причинам, связанным с безопасностью, сценарии не имеют возможности получить хранящиеся в нем URL-адреса. (Иначе любой сценарий смог бы исследовать историю посещения веб-сайтов.)

Методы **back()** и **forward()** действуют подобно кнопкам Back (Назад) и Forward (Вперед) браузера: они заставляют браузер перемещаться на один шаг назад и вперед по истории просмотра данного окна. Третий метод, **go()**, принимает целочисленный аргумент и пропускает заданное число страниц, двигаясь вперед (если аргумент положительный) или назад (если аргумент отрицательный) в списке истории.

```
history.go(-2); // Переход назад на 2 элемента,
как если бы пользователь // дважды щелкнул на
кнопке Back (Назад)
```

Современные веб-приложения способны динамически изменять содержимое страницы без загрузки нового документа. Приложениям, действующим подобным образом, может потребоваться предоставить пользователям возможность использовать

кнопки Back и Forward для перехода между этими динамически созданными состояниями приложения. На практике разработчики, когда требуется реализовать подобное управление историей просмотра, предпочитают использовать готовые решения. Многие фреймворки JavaScript включают такие решения. Например, для библиотеки jQuery существует расширение history.

Объект Navigator. Иногда сценариям бывает необходимо получить информацию о веб-браузере, в котором они выполняются. Свойство navigator объекта Window ссылается на объект Navigator, содержащий общую информацию о номере версии и о производителе броузера. Объект Navigator назван «в честь» броузера Netscape Navigator, но он также поддерживается во всех других браузерах. Объект Navigator имеет четыре свойства, предоставляющих информацию о версии работающего браузера, и вы можете использовать их для определения типа браузера:

appName Название веб-броузера. В IE это строка «Microsoft Internet Explorer». В Firefox значением этого свойства является строка «Netscape». Для совместимости с существующими реализациями определения типа браузера значением этого свойства в других браузерах часто является строка «Netscape».

appVersion Обычно значение этого свойства начинается с номера версии, за которым следует другая информация о версии браузера и его производителе. Обычно в начале строки указывается номер 4.0 или 5.0, свидетельствующий о совместимости с четвертым или пятым поколением браузеров. Формат строки в свойстве appVersion не определяется стандартом, поэтому невозможно организовать разбор этой строки способом, не зависящим от типа броузера.

userAgent Строка, которую броузер посылает в http-заголовке USER-AGENT. Это свойство обычно содержит ту же информацию, что содержится в свойстве appVersion, а также может включать дополнительные сведения. Как и в случае со свойством appVersion, формат представления этой информации не стандартизован.

Поскольку это свойство содержит больше информации, именно оно обычно используется для определения типа броузера.

platform Строка, идентифицирующая операционную систему (и, возможно, аппаратную платформу), в которой работает броузер.

Сложность свойств объекта Navigator делает невозможной универсальную реализацию определения типа браузера.

В дополнение к свойствам с информацией о версии и производителе броузера, объект Navigator имеет еще несколько свойств и методов. В число стандартных и часто реализуемых нестандартных свойств входят:

onLine Свойство navigator.onLine (если существует) определяет, подключен ли броузер к сети. Приложениям может потребоваться сохранять информацию о состоянии локально если броузер не подключен к сети.

geolocation Объект Geolocation, определяющий API для выяснения географического положения пользователя.

javaEnabled() Нестандартный метод, который должен возвращать true, если браузер способен выполнять Java-апплеты.

cookiesEnabled() Нестандартный метод, который должен возвращать true, если броузер способен сохранять cookies. Если броузер настроен на сохранение cookies только для определенных сайтов, этот метод может возвращать некорректное значение.

Свойство screen объекта Window ссылается на объект Screen, предоставляющий информацию о размере экрана на стороне пользователя и доступном количестве цветов. **Свойства width и height** возвращают размер экрана в пикселах. **Свойства availWidth и availHeight** возвращают фактически доступный размер экрана; из них исключается пространство, требуемое для таких графических элементов, как панель задач. **Свойство colorDepth** возвращает количество битов на пиксел, определяющих цвет. Типичными значениями являются 16, 24 и 32.

Свойство window.screen можно использовать, чтобы определить, не выполняется ли веб-приложение на устройстве с маленьким экраном, таком как нетбук. При ограниченном пространстве экрана, например, можно было бы использовать шрифты меньшего размера и маленькие изображения.

Сохранение данных на стороне клиента. Веб-приложения могут использовать прикладные программные интерфейсы браузеров для сохранения данных локально, на компьютере пользователя. Этот механизм сохранения данных на стороне клиента выполняет роль памяти для веб-браузеров. Веб-приложения могут сохранять, например, настройки пользователя или даже полную информацию о своем состоянии, чтобы иметь возможность возобновить работу точно с того момента, на котором их работа была прервана при последнем посещении. Хранилище на стороне клиента разграничивает данные в соответствии с происхождением, поэтому страницы с одного сайта не смогут читать данные, сохраненные страницами с другого сайта. Но две страницы с одного и того же сайта смогут использовать хранилище в качестве механизма взаимодействий. Например, данные, введенные в форме на одной странице, можно отображать в таблице на другой странице. Веб-приложения могут устанавливать срок хранения своих данных: данные могут храниться временно, т. е. получить такие данные из хранилища можно, пока не будет закрыто окно или пока броузер не завершит работу, или сохраняться на жестком диске и храниться постоянно, чтобы их можно было получить месяцы или даже годы спустя.

Web Storage. Web Storage – это прикладной программный интерфейс, определение которого первоначально было частью стандарта HTML5, но впоследствии было выделено в отдельную спецификацию. Этот прикладной интерфейс содержит объекты **localStorage и sessionStorage,** которые, по сути, являются постоянно хранимыми ассоциативными массивами, отображающими ключи в строковые значения. Интерфейс Web Storage очень прост в использовании, он подходит для хранения больших (но не огромных) объемов данных и доступен во всех текущих браузерах, но не поддерживается старыми браузерами.

Объекты localStorage и sessionStorage. Браузеры, реализующие положения проекта спецификации «Web Storage», определяют в объекте Window два свойства: localStorage и sessionStorage. Оба свойства ссылаются на объект Storage – постоянно хранимый ассоциативный массив, отображающий строковые ключи в строковые значения. Объекты Storage действуют подобно обычным объектам в языке JavaScript: достаточно просто присвоить свойству объекта строку, и броузер автоматически сохранит

ее. Разница между localStorage и sessionStorage заключается лишь в сроке хранения и области видимости: они определяют, как долго будут храниться данные и кому они будут доступны.

Следующий фрагмент использует свойство localStorage, но он точно так же мог бы работать и со свойством sessionStorage:

```
var name = localStorage.username; // Получить
сохраненное значение.

name = localStorage["username"]; // Эквивалентная форма
обращения, как к массиву

if (!name) {
      name = prompt("Как вас зовут?"); // Задать
пользователю вопрос.
      localStorage.username = name; // Сохранить ответ.
}

// Выполнить итерации по всем хранящимся
парам имя/значение
for(var name in localStorage) { // Итерации по всем
хранящимся именам
      var value = localStorage[name]; // Получить
значение для каждого из них
}
```

Проект спецификации «Web Storage» определяет возможность сохранения структурированных данных (объектов и массивов), а также простых значений и данных встроенных типов, таких как даты, регулярные выражения и даже объекты File. Если потребуется сохранять и извлекать данные таких типов, их можно кодировать и декодировать вручную. Например:

```
// При сохранении числа оно автоматически
преобразуется в строку.
// Не забудьте выполнить обратное
преобразование при извлечении из хранилища.

localStorage.x = 10;
var x = parseInt(localStorage.x);

// Для кодирования любых простых или
структурированных данных удобно
// использовать формат JSON

localStorage.data = JSON.stringify(data); // Закодировать и
сохранить
var data = JSON.parse(localStorage.data); // Извлечь и
декодировать.
```

Срок хранения и область видимости. Объекты localStorage и sessionStorage отличаются сроком хранения данных и областью видимости хранилища. Объект localStorage представляет долговременное хранилище данных: срок хранения не

ограничен, и данные сохраняются на компьютере пользователя, пока не будут удалены веб-приложением или пока пользователь не потребует от броузера (посредством некоторого пользовательского интерфейса, предоставляемого браузером) удалить их.

Доступность данных в объекте localStorage ограничивается происхождением документа. Происхождение документа определяется такими параметрами, как протокол, имя хоста и номер порта, поэтому все следующие URL-адреса ссылаются на документы с разным происхождением:

```
http://www.example.com // Протокол: http; имя хоста: www.example.com
https://www.example.com // Другой протокол
http://static.example.com // Другое имя хоста
http://www.example.com:8000 // Другой порт
```

Все документы, имеющие одно и то же происхождение, будут совместно использовать одни и те же данные в объекте localStorage (независимо от происхождения сценария, который фактически обращается к хранилищу localStorage). Они смогут читать и изменять данные друг друга. Но документы с разными происхождениями никогда не смогут прочитать или изменить данные друг друга (даже если оба они будут выполнять сценарий, полученный с одного и того же стороннего сервера).

Обратите внимание, что видимость данных в хранилище localStorage также ограничивается конкретным браузером. Если посетить сайт с помощью Firefox, а затем вновь посетить его, например, с помощью Chrome, никакие данные, сохраненные при первом посещении, не будут доступны при втором посещении.

Данные, сохраняемые в sessionStorage, имеют другой срок хранения: они хранятся, пока остается открытым окно верхнего уровня или вкладка браузера, в которой выполнялся сценарий, сохранивший эти данные. При закрытии окна или вкладки все данные, хранящиеся в sessionStorage, удаляются. (Отметьте, однако, что современные браузеры имеют возможность повторно открывать недавно закрытые вкладки и восстанавливать последний сеанс работы с браузером, поэтому срок хранения информации об этих вкладках и связанных с ними хранилищах sessionStorage может оказаться больше, чем кажется.)

Доступность данных в хранилище sessionStorage, как и в хранилище localStorage, ограничивается происхождением документа, т.е. документы с разным происхождением никогда не смогут совместно использовать одни и те же данные в sessionStorage. Но помимо этого доступность данных в хранилище sessionStorage ограничивается также окном. Если пользователь откроет в браузере две вкладки, отображающие документы с общим происхождением, эти две вкладки будут владеть разными хранилищами sessionStorage.

Прикладной программный интерфейс объекта Storage. Объекты localStorage и sessionStorage часто используются как обычные объекты языка JavaScript: присваивание значения свойству приводит к сохранению строки, а чтение свойства – к ее извлечению из хранилища. Но эти объекты определяют также более формальный прикладной интерфейс, основанный на методах. Сохранить значение можно с помощью метода **setItem(),** передав ему имя и значение. Извлечь значение можно с помощью метода **getItem()**, передав ему имя. Удалить значение можно с помощью метода **removeItem(),** передав ему имя. (В большинстве браузеров удалить значение можно также с помощью

оператора delete, как если бы оно было обычным объектом.) Удалить все хранящиеся значения можно вызовом метода **clear()** (без аргументов). Наконец, перечислить имена всех хранящихся значений можно с помощью свойства **length** и метода **key(),** передавая ему значения от 0 до length-1. Ниже приводится несколько примеров использования объекта localStorage. Этот программный код с тем же успехом мог бы использовать объект sessionStorage:

```
    localStorage.setItem("x", 1); // Сохранить число под именем "x"
    localStorage.getItem("x"); // Извлечь значение

    // Перечислить все хранящиеся пары имя-значение
    for(var i = 0; i < localStorage.length; i++) { // length дает количество пар
        var name = localStorage.key(i); // Получить имя i-й пары
        var value = localStorage.getItem(name); // Получить значение этой пары
    }

    localStorage.removeItem("x"); // Удалить элемент "x"

    localStorage.clear(); // Удалить все остальные элементы
```

Несмотря на то что обычно удобнее сохранять и извлекать данные, обращаясь к свойствам, тем не менее иногда может потребоваться использовать эти методы. Во-первых, метод clear() не имеет эквивалента и является единственным способом удаления всех пар имя/значение в объекте Storage. Аналогично метод removeItem() является единственным переносимым способом удаления одной пары имя/значение.

События объекта Storage. При изменении данных, хранящихся в localStorage или sessionStorage, браузер генерирует событие «storage» во всех объектах Window, в которых доступны эти данные (но не в окне, где выполнялось сохранение). Если в браузере открыты две вкладки со страницами с общим происхождением и в одной из страниц производится сохранение значения в localStorage, в другой вкладке будет сгенерировано событие «storage». Обратите также внимание, что события «storage» генерируются, только когда содержимое хранилища действительно изменяется. Присваивание хранимому элементу его текущего значения, как и попытка удалить несуществующий элемент, не возбуждают событие. Регистрация обработчиков события «storage» выполняется с помощью метода addEventListener(). В большинстве браузеров для этой цели можно также использовать **свойство onstorage объекта Window.** Наконец, обратите внимание, что объект localStorage и событие «storage» могут служить широковещательным механизмом, с помощью которого броузер может отправлять сообщения всем окнам, в которых в настоящий момент открыт один и тот же веб-сайт. В качестве другого примера представьте веб-приложение графического редактора, позволяющее пользователю отображать палитры с инструментами в отдельных окнах. При выборе пользователем некоторого инструмента приложение могло бы сохранять в localStorage признак выбранного инструмента и тем самым рассылать другим окнам извещения о том, что был выбран новый инструмент.

Задание на лекции:
1. Создать проект с базовой структурой
2. Разработать авторизацию
3. Дополнить авторизацию регистрацией
4. Защитить страницу от прямого доступа.
3. Написать код для отображения часов.

Домашнее задание:
1. Создайте отсчет от 0 до бесконечности.
2. Создайте отсчет с кнопкой остановки.
3. Создайте таймер обратного отсчета с кнопкой старта.
4. Создайте простой слайдер с кнопкой старт