***24 марта 2020 г.* Лекция по дисциплине «WEB-программирование»**

Конспект и основные тезисы:
файл «WEB(лк) - 05.01. JavaScript - Введение в JS.pdf»

Теория:

Введение в Javascript

JavaScript – это язык программирования для Веб. Подавляющее большинство веб-сайтов используют JavaScript, и все современные веб-браузеры – для настольных компьютеров, игровых приставок, электронных планшетов и смартфонов – включают интерпретатор JavaScript, что делает JavaScript самым широкоприменимым языком программирования из когда-либо существовавших в истории.

JavaScript входит в тройку технологий, которые должен знать любой веб-разработчик: язык разметки HTML, позволяющий определять содержимое веб-страниц, язык стилей CSS, позволяющий определять внешний вид веб-страниц, и язык программирования JavaScript, позволяющий определять поведение веб-страниц.

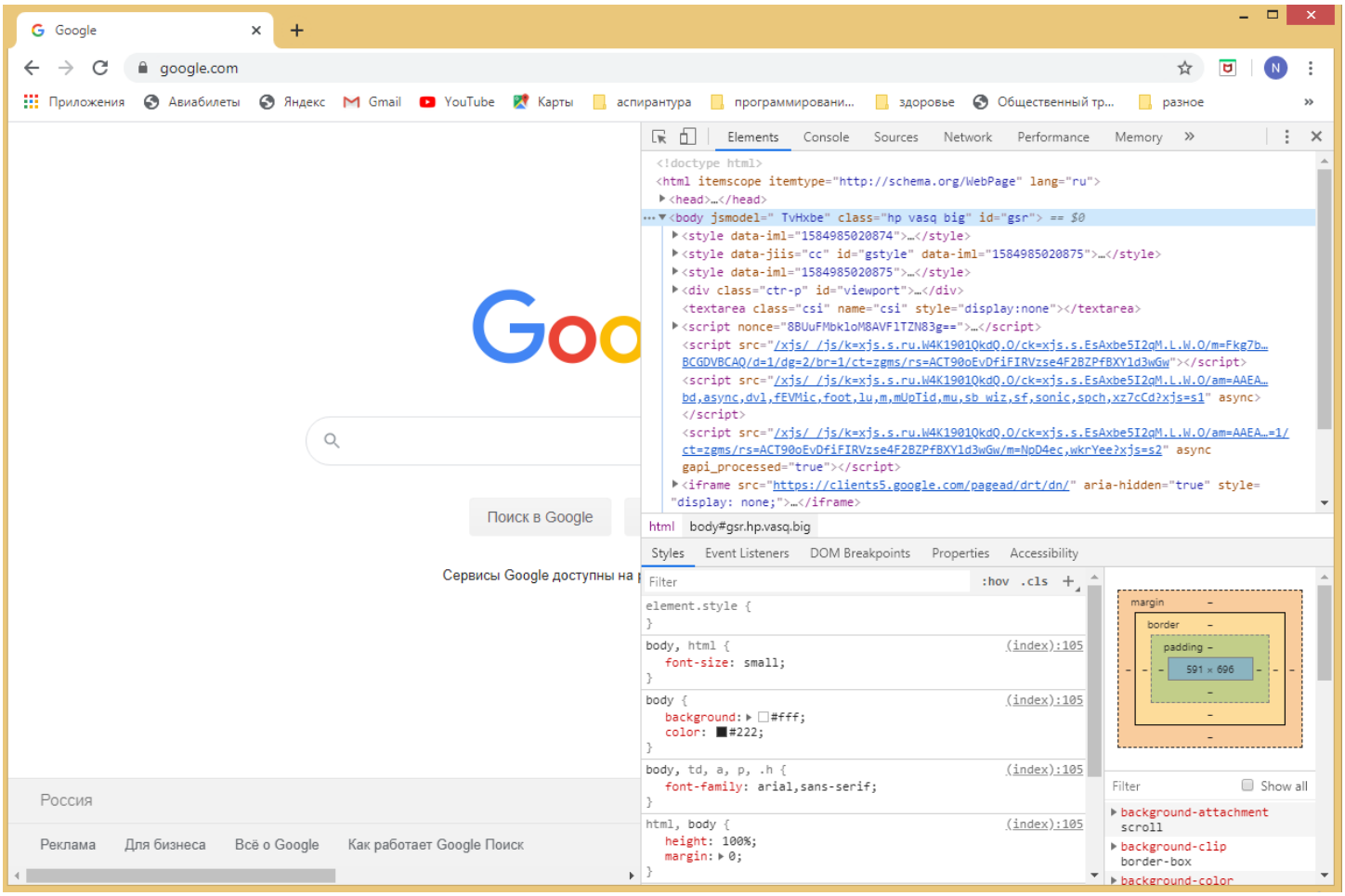
JavaScript является высокоуровневым, динамическим, нетипизированным и интерпретируемым языком программирования, который хорошо подходит для программирования в объектно-ориентированном и функциональном стилях. Свой синтаксис JavaScript унаследовал из языка Java, свои функции – из языка Scheme, а механизм наследования на основе прототипов – из языка Self.

Название языка «JavaScript» может вводить в заблуждение. За исключением поверхностной синтаксической схожести, JavaScript полностью отличается от языка программирования Java. JavaScript давно перерос рамки языка сценариев, превратившись в надежный и эффективный универсальный язык программирования. Последняя версия языка (смотрите врезку) определяет множество новых особенностей, позволяющих использовать его для разработки крупномасштабного программного обеспечения.

Чтобы представлять хоть какой-то интерес, каждый язык программирования должен иметь свою платформу, или стандартную библиотеку, или API функций для выполнения таких базовых операций, как ввод и вывод. Ядро языка JavaScript определяет минимальный прикладной интерфейс для работы с текстом, массивами, датами и регулярными выражениями, но в нем отсутствуют операции ввода-вывода. Ввод и вывод (а также более сложные возможности, такие как сетевые взаимодействия, сохранение данных и работа с графикой) перекладываются на «окружающую среду», куда встраивается JavaScript. Обычно роль окружающей среды играет веб-браузер.

Исследование JavaScript

Изучая новый язык программирования, очень важно стараться пробовать **запускать примеры**, изменять их и опять запускать, чтобы проверить, насколько правильно вы понимаете особенности языка. Для этого необходим интерпретатор JavaScript. Любой веб-браузер включает интерпретатор JavaScript. Обычно эти инструменты можно отыскать в меню Tools (Инструменты или Сервис) браузера в виде пункта Developer-Tools (Средства разработчика). Например, у браузера Chrome доступ к средствам разработки и отладки можно получить при нажатии на F12 (см. рис.).



Панель или окно типичного «инструмента разработчика» включает множество вкладок, позволяющих исследовать структуру HTML-документа, стили CSS, наблюдать за выполнением сетевых запросов и т. д. Среди них имеется вкладка Console (Консоль), где можно вводить строки программного кода JavaScript и выполнять их. Это самый простой способ поэкспериментировать с JavaScript.

В современных браузерах имеется простой переносимый API консоли. Для вывода текста в консоль можно использовать функцию console.log(). Зачастую такая возможность оказывается удивительно полезной при отладке. Похожий, но более навязчивый способ вывода информации или отладочных сообщений заключается в передаче строки текста функции alert(), которая отображает его в окне модального диалога.

Что такое JavaScript?

Краткая история JavaScript. Реализации JavaScript. ECMAScript. Понятие DOM. Уровни DOM. Другие DOM. – читай файл «*WEB – JS. Введение.pdf*». (по книге Закас Н. JavaScript для профессиональных веб-разработчиков / [Пер. с англ. А. Лютича]. -СПб.: Питер, 2015. -960 с.: ил. - (Серия «Для профессионалов»)).

Объектная модель браузера

Объектная модель браузера (Browser Object Model, BOM), которая обеспечивает доступ к окну браузера и позволяет манипулировать его элементами. Используя BOM, можно взаимодействовать с браузером вне контекста отображаемой страницы. До недавних пор BOM была единственной частью реализации JavaScript, не имеющей стандарта, из-за чего при работе с ней часто возникали проблемы. Формализация многих элементов BOM в HTML5 изменила ситуацию к лучшему, прояснив многие неясные аспекты модели.

BOM регламентирует работу с окном и фреймами браузера, но любое специфичное для браузера javaScript-расширение тоже обычно считается частью BOM. Вот некоторые такие расширения:

- функция отображения всплывающих окон в браузере;
- возможность перемещать, закрывать и изменять размеры окна браузера;
- объект navigator, предоставляющий подробные сведения о браузере;
- объект location, предоставляющий подробные сведения о странице, загруженной в браузере;
- объект sc reen, предоставляющий подробные сведения о разрешении экрана;
- поддержка cookie-файлов;
- настраиваемые объекты, включая XMLHttpRequest, а также ActiveXObject в Internet Explorer.

Стандартов БОМ долго не было, поэтому в каждом браузере она реализована посвоему

JavaScript в HTML

Элемент <script>. Расположение тегов. Отложенные сценарии. Асинхронные сценарии. Устаревший синтаксис. – читай файл «*WEB – JS. JavaScript в HTML.pdf*». (по книге Закас Н. JavaScript для профессиональных веб-разработчиков / [Пер. с англ. А. Лютича]. -СПб.: Питер, 2015. -960 с.: ил. - (Серия «Для профессионалов»)).

Синхронные, асинхронные и отложенные сценарии (продолжение)

Когда поддержка JavaScript впервые появилась в веб-броузерах, не существовало никаких инструментов обхода и управления структурой содержимого документа. Единственный способ, каким JavaScript-код мог влиять на содержимое документа, - это генерировать содержимое в процессе загрузки документа. Делалось это с помощью метода document.write(). В примере далее показано, как выглядел ультрасовременный JavaScript-код в 1996 году.

Пример. Генерация содержимого документа во время загрузки

```
<h1>Таблица факториалов</h1>
<script>
   function factorial(n) { // Функция вычисления факториалов
      if (n <= 1) return n;
          else return n*factorial(n-1);
   }
   // Начало HTML-таблицы
   document.write("<table>");
   // Вывести заголовок таблицы
   document.write("<tr><th>n</th><th>n!</th></tr>");
   for(var i = 1; i <= 10; i++) { // Вывести 10 строк
      document.write("<tr><td>" + i + "</td><td>" +
          factorial(i) + "</td></tr>");
   }
   document.write("</table>"); // Конец таблицы
   document.write("Generated at " + new Date()); // Вывести время
</script>
```

Когда сценарий передает текст методу document.write(), этот текст добавляется во входной поток документа, и механизм синтаксического анализа разметки HTML действует так, как если бы элемент <script> был замещен этим текстом. Использование метода document.write() более не считается хорошим стилем программирования, но его применение по-прежнему возможно, и этот факт имеет важное следствие. Когда механизм синтаксического анализа разметки HTML встречает элемент <script>, он должен, по умолчанию, выполнить сценарий, прежде чем продолжить разбор

и отображение документа. Это не является проблемой для встроенных сценариев, но если сценарий находится во внешнем файле, на который ссылается атрибут src, это означает, что часть документа, следующая за сценарием, не появится в окне браузера, пока сценарий не будет загружен и выполнен.

Такой *синхронный*, или *блокирующий*, порядок выполнения действует только по умолчанию. Тег <script> может иметь атрибуты defer и async, которые (в браузерах, поддерживающих их) определяют иной порядок выполнения сценариев. Это логические атрибуты - они не имеют значения; они просто должны присутствовать в теге <script>. Согласно спецификации HTML5, эти атрибуты принимаются во внимание, только когда используются вместе с атрибутом src, однако некоторые браузеры могут поддерживать атрибут defer и для встроенных сценариев:

<script defer src= "deferred. js"></script>

<script async src="async.js"></script>

Оба атрибута, defer и async, сообщают броузеру, что данный сценарий не использует метод document.writeO и не генерирует содержимое документа, и что броузер может продолжать разбор и отображение документа, пока сценарий загружается. Атрибут defer заставляет броузер отложить выполнение сценария до момента, когда документ будет загружен, проанализирован и станет готов к выполнению операций. Атрибут async заставляет броузер выполнить сценарий, как только это станет возможно, но не блокирует разбор документа на время загрузки сценария.

Если тег <script> имеет оба атрибута, броузер, поддерживающий оба этих атрибута, отдаст предпочтение атрибуту async и проигнорирует атрибут defer. Обратите внимание, что отложенные сценарии выполняются в порядке их следования в документе. Асинхронные сценарии выполняются сразу же, как только будут загружены, т. е. они могут выполняться в произвольном порядке.

Выполнение JavaScript-программ

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.

Если веб-страница содержит встроенный фрейм (элемент <iframe>), JavaScript-код во встроенном документе будет работать с другим глобальным объектом, отличным от глобального объекта в объемлющем документе, и его можно рассматривать как отдельную JavaScript-программу.

URL-адреса javascript: в букмарклетах существуют за пределами какого-либо документа, и их можно рассматривать, как своего рода пользовательские расширения или дополнения к другим программам. Когда пользователь запускает букмарклет, программный код в букмарклете получает доступ к глобальному объекту и содержимому текущего документа и может манипулировать им как угодно.

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 (но не всегда) 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-браузер вызывает функции обработчиков (которые определены в HTML-атрибутах обработчиков событий, установлены сценариями, выполненными на первом этапе, или обработчиками событий, вызывавшимися ранее) 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. 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.

Обе разновидности языка, базовый JavaScript и клиентский JavaScript, поддерживают однопоточную модель выполнения. Сценарии и обработчики событий выполняются последовательно, не конкурируя друг с другом. Такая модель выполнения обеспечивает простоту программирования на языке JavaScript.

Последовательность выполнения клиентских сценариев

Идеализированная последовательность выполнения JavaScript-программ в браузерах следующая:

1. Веб-броузер создает объект Document и начинает разбор веб-страницы, добавляя в документ объекты Element и текстовые узлы в ходе синтаксического анализа HTML-элементов и их текстового содержимого. На этой стадии свойство document. readyState получает значение «loading».

2. Когда механизм синтаксического анализа HTML встречает элементы <script>, не имеющие атрибута async и/или defer, он добавляет эти элементы в документ и затем выполняет встроенные или внешние сценарии. Эти сценарии выполняются синхронно, а на время, пока сценарий загружается (если это необходимо) и выполняется, синтаксический анализ документа приостанавливается. Такие сценарии могут использовать метод document.write() для вставки текста во входной поток. Этот текст станет частью документа, когда синтаксический анализ продолжится. Синхронные сценарии часто просто определяют функции и регистрируют обработчики событий для последующего использования, но они могут исследовать и изменять дерево документа, доступное на момент их запуска. То есть синхронные сценарии могут видеть собственный элемент <script> и содержимое документа перед ним.

3. Когда механизм синтаксического анализа встречает элемент <script>, имеющий атрибут async, он начинает загрузку сценария и продолжает разбор документа. Сценарий будет выполнен

сразу же по окончании его загрузки, но синтаксический анализ документа не приостанавливается на время загрузки сценария. Асинхронные сценарии не должны использовать метод document.write(). Они могут видеть собственный элемент <script>, все элементы документа, предшествующие ему и, возможно, дополнительное содержимое документа.

4. По окончании анализа документа значение свойства document. readyState изменяется на «interactive».

5. Выполняются все сценарии, имеющие атрибут defer, в том порядке, в каком они встречаются в документе. В этот момент также могут выполняться асинхронные сценарии. Отложенные сценарии имеют доступ к полному дереву документа и не должны использовать метод document. write().

6. Браузер возбуждает событие «DOMContentLoaded» в объекте Document. Это событие отмечает переход от этапа синхронного выполнения сценариев к управляемому событиями асинхронному этапу выполнения программы. Следует, однако, отметить, в этот период также могут выполняться асинхронные сценарии, которые не были еще выполнены.

7. К этому моменту синтаксический анализ документа завершен, но браузер все еще может ожидать окончания загрузки дополнительного содержимого, такого как изображения. Когда все содержимое будет загружено и все асинхронные сценарии будут выполнены, свойство document. readyState получит значение «complete» и веб-броузер возбудит событие «load» в объекте Window.

8. С этого момента будут асинхронно вызываться обработчики событий в ответ на действия пользователя, сетевые операции, истечение таймера и т. д.

Дополнительно по теме:

Флэнаган Дэвид. JavaScript. Подробное руководство. 6-е изд. – Пер. с англ. – СПб: Символ-Плюс, 2012. – 1080с., ил.:

Часть 13.2. Встраивание JavaScript-кода в разметку HTML:

Часть 13.2.1. Элемент <script> - с. 337

Часть 13.2.2. Сценарии во внешних файлах – с. 338

Часть 13.2.4. Обработчики событий в HTML – с. 340

Часть 13.2.5. JavaScript в URL – с. 341