**Лабораторная работа 1.**

**Программное взаимодействие с HTML документами на основе DOM API.**

**Цель работы:** Знакомство с языком разработки клиентских веб-сценариев JavaScript. Применение JavaScript для автоматизации процесса разметки и добавления интерактивных возможностей веб-страниц.

**Теоретический материал**

1 Элементы языка JavaScript

JavaScript позволяет "оживить" веб-страницу. Это реализуется путем добавления к статическому описанию фрагмент исполняемого кода. JavaScript-сценарий может взаимодействовать с любыми компонентами HTML-документа и реагировать на изменение их состояния.

JavaScript не является строго типизированным языком, в переменных могут храниться практически любые типы данных.

Как и программа на языке Java, сценарий JavaScript выполняется под управлением интерпретатора. Однако если Java-приложение или Java-аплет компилируется в байтовый код, то сценарий JavaScript интерпретируется на уровне исходного текста.

Следует отметить, что языковые конструкции JavaScript совпадают с соответствующими средствами C++ и Java.

1.1.Структура сценария

*Сценарием JavaScript* считается фрагмент кода, расположенный между дескрипторами <SCRIPT> и </SCRIPT>:

*Текст HTML-документа*

*<SCRIPT>*

*Код сценария*

*</SCRIPT>*

*Текст HTML-документа*

1.2.Переменные

В сценариях JavaScript переменные могут хранить данные любых типов: числа, строки текста, логические значения, ссылки на объекты, а также специальные величины, например "нулевое" значение *null* или значение *NaN*, которое сообщает о недопустимости операции.

Переменная в языке JavaScript объявляется с помощью ключевого слова **var**. Так, например, выражение

**var** *selected = "first item";*

создает переменную с именем select и присваивает ей в качестве значения строку символов "*first item*". Переменные могут объявляться также автоматически. Это происходит при присвоении значения переменной, не встречавшейся ранее в данном сценарии. Так, в следующем примере создается переменная с именем *rating*, которой присваивается числовое значение, равное *512.5*:

*rating = 512.5;*

1.3.Объекты

В языке JavaScript не предусмотрены средства для работы с классами в том виде, в котором они реализованы в C++ или Java. Разработчик сценария не может создать подкласс на основе существующего класса, переопределить метод или выполнить какую-либо другую операцию с классом. Сценарию, написанному на языке JavaScript, в основном доступны лишь готовые объекты. Построение нового объекта приходится выполнять лишь в редких случаях.

Объекты содержат *свойства* (свойства объектов можно сравнить с переменными) и *методы*. Объекты, а также их свойства и методы идентифицируются *именами*. Объектами являются формы, изображения, гипертекстовые ссылки и другие компоненты веб-страницы, HTML-документ, отображаемый в окне браузера, окно браузера и даже сам браузер. В процессе работы JavaScript сценарий обращается к этим объектам, получает информацию и управляет ими.

Кроме того, разработчику сценария на языке JavaScript доступны объекты, не связанные непосредственно с HTML-документом. Их называют *предопределенными,* или *независимыми* объектами. С помощью этих объектов можно реализовать массив, описать дату и время, выполнить математические вычисления и решить некоторые другие задачи.

Первый объект, с которым необходимо познакомиться, чтобы написать простейший сценарий, - это объект *document*, который описывает HTML документ, отображаемый в окне браузера. Ниже приведен исходный текст веб-страницы, содержащей сценарий, действия которого сводятся к выводу строки текста в окне браузера.

|  |
| --- |
| <HTML>  <НЕАD>  <ТIТLЕ>Первый сценарий JavaScript</TITLE></HEAD>  <BODY>  <SCRIPT LANGUAGE="JavaScript">  document.write("Проверка сценария JavaScript");  </SCRIPT>  </BODY>  </HTML> |

Имена чувствительны к регистрам символов, и если вы попытаетесь обратиться к текущему документу по имени *Document*, интерпретатор JavaScript отобразит сообщение об ошибке.

Основное назначение сценариев JavaScript - создавать динамически изменяющиеся объекты, корректиро­вать содержимое HTML-документов в зависимости от особенностей окружения, осу­ществлять взаимодействие с пользователем и т.д.

1.4.Операции

Набор операторов в JavaScript, их назначение и правила использования в основном совпадают с принятыми в языке С++. Исключением является операция задаваемая символом "+".

В JavaScript символ "+" определяет как *суммирование* числовых значений, так и *конкатенацию* строк.

Так, например, в результате вычисления выражения

*sum = 47 + 21;*

переменной *sum* будет присвоено значение *68*, а после выполнения операции

*sum = "строка 1 " + "строка 2";*

в переменную *sum* будет записана последовательность символов "*строка 1 строка 2*".

Рассмотрим еще один пример:

|  |
| --- |
| <HTML>  <BODY>  <H2>Числа и строки</H2><BR>  <SCRIPT LANGUAGE="JavaScript">  var a = 3;  var b = 8;  var c = " попугаев ";  document.write("a+b="); document.write(a + b); document.write("<BR>");  document.write( "a + с = "); document.write(a+c);  document.write("<BR>");  document.write("c + a = "); document.write (c + a);  document.write ("<BR>");  document.write ("a + b + с = "); document.write(a + b + c);  document.write("<BR>");  document.write("c + a + b = "); document.write(c + a + b);  document.write("<BR>");  </SCRIPT>  </BODY>  </HTML> |

В окне браузера приведенный выше HTML-код выглядит так, как показано на скриншоте

|  |
| --- |
| JSString |

Первая строка отображает результат суммирования двух числовых значений, вторая и третья - результат конкатенации строки и символьного представления числа. Если операция суммирования чисел предшествует конкатенации, JavaScript вычисляет сумму чисел, представляет ее в символьном виде, затем производит конкатенацию двух строк. Если же первой в выражении указана операция конкатенации, то JavaScript сначала преобразует числовые значения в символьный вид, а затем выполняет конкатенацию строк.

1.5.Управляющие конструкции

Управляющие конструкции, используемые в языке С++, в основном применимы и в сценариях JavaScript.

В JavaScript дополнительно определены языковые конструкции, отсутствующие в С++, а именно: операторы *for...in* и *with*.

В *примере 1* с помощью оператора цикла на веб-странице формируется таблица умножения чисел.

Пример 1.

|  |
| --- |
| <html>  <body>  <table>  <script language="JavaScript">  document.write("<tr><td>&nbsp;</td>");  for (i = 1; i < 10; i++) document.write("<td>"+i+"&nbsp;</td>");  document.write("</tr>");  for (i = 1; i < 10; i++)  {  document.write("<tr><td>" + i + "&nbsp;</td>");  for (j = 1; j < 10; j++)  {  document.write("<td bgcolor='#00ffa0'>" + (i\*j) + "&nbsp;</td>"); }  document.write("</tr>");  }  </script>  </table>  </body>  </html> |

Отдельного внимания заслуживает оператор *new*. Несмотря на то, что большинство объектов уже созданы браузером и доступны сценарию, в некоторых случаях приходится создавать объекты в процессе работы. Это относится к предопределенным объектам и объектам, определяемым разработчиком сценария. Для создания объекта используется оператор *new*, который вызывается следующим образом:

*переменная =* ***new*** *тип\_ объекта (параметры)*

1.6.Функции

Формат объявления функции выглядит следующим образом:

***function*** *имя функции ([ параметры]) тело функции*

Объявление функции начинается с ключевого слова **function**. Так же, как и в языке *С* для идентификации функции используется имя, при вызове функции могут передаваться параметры, а по окончании выполнения возвращаться значение. Однако, в отличие от *С*, тип возвращаемого значения и типы параметров не за­даются. Ниже показаны два способа вызова функции

* *имя\_функции ([параметры]);*
* *переменная = имя функции ([параметры]);*

Во втором случае значение, возвращаемое функцией, присваивается указанной пе­ременной.

1.7.Область видимости переменных

Работа с переменными в теле функции подчиняется следующим правилам.

* Если переменная объявлена с помощью ключевого слова *var*, доступ к ней осуществляется по правилам, подобным тем, которые используются в языке *С*.
* Переменная, объявленная внутри функции, считается *локальной*. Область видимости такой переменной ограничивается телом функции, в которой она объявлена.
* Переменная, объявленная вне функции, считается *глобальной*. К ней можно обращаться из любой точки сценария.
* Если локальная и глобальная переменные имеют одинаковые имена, то в теле функции локальная переменная "маскирует" глобальную.
* Если переменная создается автоматически, т.е. если она не объявлена с помо­щью ключевого слова *var*, но присутствует в левой части оператора прямого присваивания, то она считается глобальной и становится доступной из любой точки сценария.

1. **HTML DOM**

*DOM* (Document Object Model) – представляет собой стандарт консорциума W3C для программного доступа к документам HTML или XML. Фактически это платформо- и языково-нейтральный интерфейс, позволяющий программам и сценариям динамически обращаться и обновлять содержимое, структуру и стиль документа.

В рамках данного стандарта можно выделить 3 части:

• Core DOM – стандартная модель любого структурированного документа

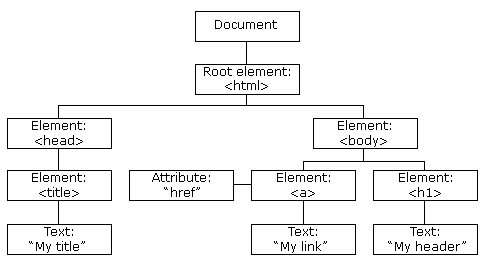
• XML DOM - стандартная модель XML документа

• HTML DOM - стандартная модель HTML документа

*DOM* определяет объекты и свойства всех элементов документа и методы (интерфейс) для доступа к ним.

*HTML DOM* определяет объекты и свойства всех HTML элементов и методы (интерфейс) для доступа к ним. Иначе говоря, HTML DOM описывает каким образом необходимо получать, изменять, добавлять и удалять HTML элементы.

В соответствии с моделью DOM все, что содержится внутри HTML документа - является узлом. То есть HTML документ представляется в виде дерева узлов, которыми являются элементы, атрибуты и текст.



**2.1.Узлы дерева HTML документа**

Согласно модели DOM:

* Весь документ представляется узлом документа;
* Каждый HTML тэг является узлом элемента;
* Текст внутри HTML элементов представляется текстовыми узлами;
* Каждому HTML атрибуту соответствует узел атрибута;
* Комментарии являются узлами комментариев.

Пример 2.

|  |
| --- |
| <html>  <head>  <title>HTML документ</title>  </head>  <body>  <h1>Заголовок </h1>  <p>Просто текст</p>  </body>  </html> |

В этом примере корневым узлом является тэг <html>. Все остальные узлы содержатся внутри <html>. У этого узла имеется два дочерних узла: <head> и <body>. Узел <head> содержит узел <title>, а узел <body> содержит узлы <h1> и <p>.

Следует обратить особое внимание на то, что текст, расположенный в узле элемента соответствует текстовому узлу. В примере ***<title>HTML документ</title>*** узел элемента <title> содержит текстовый узел «*HTML документ*», то есть «*HTML документ*» не является значением элемента <title>. Тем не менее, в рамках HTML DOM значение текстового узла может быть доступно посредством свойства *innerHTML*.

Все узлы HTML документа могут быть доступны посредством дерева, при этом их содержимое может быть изменено или удалено, а также можно добавить новые элементы.

Все узлы дерева находятся в иерархических отношениях между собой. Для описания этих отношений используются термины *родитель*, *дочерний* *элемент* и *потомок*. Родительские узлы имеют дочерние узлы, а дочерние элементы одного уровня называются потомками (братьями или сестрами).

В отношении узлов дерева соблюдаются следующие принципы:

* Самый верхний узел дерева называется корневым;
* Каждый узел, за исключением корневого, имеет ровно один родительский узел;
* Узел может иметь любое число дочерних узлов;
* Конечный узел дерева не имеет дочерних узлов;
* Потомки имеют общего родителя.

**2.2.Программный интерфейс HTML DOM**

В рамках DOM модели HTML можно рассматривать как множество узловых *объектов*. Доступ к ним осуществляется с помощью *JavaScript* или других языков программирования. Программный интерфейс DOM включает в себя набор стандартных *свойств* и *методов*.

*Свойства* представляют некоторые сущности (например, <h1>), а *методы* - действия над ними (например, *добавить <a>*).

К типичным свойствам DOM относятся следующие:

* x.*innerHTML* – внутреннее текстовое значение HTML элемента **x**;
* x.*nodeName* – имя **x**;
* x.*nodeValue* – значение **x**;
* x.*parentNode* – родительский узел для **x**;
* x.*childNodes* – дочерний узел для **x**;
* x.*attributes* – узлы атрибутов **x**.

Узловой объект, соответствующий HTML элементу поддерживает следующие методы:

* x.*getElementById*(id) – получить элемент с указанным **id**;
* x.*getElementsByTagName*(name) – получить все элементы с указанным именем тэга (**name**);
* x.*appendChild*(node) – вставить дочерний узел для **x**;
* x.*removeChild*(node) – удалить дочерний узел для **x**.

Пример 3.

Для получения текста из элемента *<p>* со значением атрибута *id* "*demo*" в HTML документе можно использовать следующий код:

*txt = document.getElementById("demo").innerHTML*

Тот же самый результат может быть получен по-другому:

*txt=document.getElementById("demo").childNodes[0].nodeValue*

В рамках DOM возможны 3 способа доступа к узлам:

1. С помощью метода *getElementById*(ID). При этом возвращается элемент с указанным **ID**.
2. С помощью метода *getElementsByTagName*(name). При этом возвращаются все узлы с указанным именем тэга (в виде индексированного списка). Первый элемент в списке имеет нулевой индекс.
3. Путем перемещения по дереву с использованием отношений между узлами.

Для определения длины списка узлов используется свойство *length*.

Пример 4.

|  |
| --- |
| x = document.*getElementsByTagName*("p");  for (i = 0; i < x.*length*; i++)  {  document.*write*(x[i].*innerHTML*);  document.*write*("<br/>");  } |

В данном примере внутрь HTML документа вставляется в виде списка текстовое содержимое всех элементов соответствующих тэгу <p>.

Для навигации по дереву в ближайших окрестностях текущего узла можно использовать следующие свойства:

* *parentNode;*
* *firstChild;*
* *lastChild*.

Для непосредственного доступа к тэгам можно использовать 2 специальных свойства:

* document.*documentElement* – для доступа к корневому узлу документа;
* document.*body* – для доступа к тэгу <*body*>.

**2.3.Свойства узлов**

В HTML DOM каждый узел является объектом, который может иметь методы (функции) и свойства. Наиболее важными являются следующие свойства:

* *nodeName*;
* *nodeValue*;
* *nodeType*.

Свойство *nodeName* указывает на имя узла. Это свойство имеет следующие особенности:

* Свойство *nodeName* предназначено только для чтения;
* Свойство *nodeName* узла элемента точно соответствует имени тэга;
* Свойство *nodeName* узла атрибута соответствует имени атрибута;
* Свойство *nodeName* текстового узла всегда равно **#text**
* Свойство *nodeName* узла документа всегда равно **#document**

**Замечание:** *nodeName* всегда содержит имя тэга HTML элемента в верхнем регистре.

Свойство *nodeValue* указывает на значение узла. Это свойство имеет следующие особенности:

* Свойство *nodeValue* узла элемента не определено;
* Свойство *nodeValue* текстового узла указывает на сам текст;
* Свойство *nodeValue* узла атрибута указывает на значение атрибута.

Свойство *nodeType* возвращает тип узла. Это свойство предназначено только для чтения:

Наиболее важными типами узлов являются следующие:

|  |  |
| --- | --- |
| **Тип элемента** | **Тип узла** |
| Element | 1 |
| Attribute | 2 |
| Text | 3 |
| Comment | 8 |
| Document | 9 |

**2.4.Изменение HTML элементов**

HTML элементы могут быть изменены с посредством использования JavaScript, HTML DOM и событий.

В примере 5 показано, как можно динамически изменять текстовое содержимое тэга <p>:

Пример 5.

|  |
| --- |
| <html>  <body>  <p id="p1">Hello World!</p>  <script type="text/javascript">  document.*getElementById*("p1").*innerHTML*="New text!";  </script>  </body>  </html> |

1. **Диалоговые элементы**

В JavaScript поддерживается работа со следующими диалоговыми элементамиинтерфейса:

1. **Alert**. Применяется для уведомления пользователя, работающего с веб-браузером.

Синтаксис:

*alert("сообщение");*

1. **Confirm**. Применяется для выбора пользователем одного из двух вариантов ответа «Да/Нет». Соответственно *Confirm* возвращает значение *true/false*.

Синтаксис:

*confirm("вопрос");*

1. **Prompt**. Применяется для ввода пользователем значения. При нажатии «OK» возвращается введенное значение, в случае «Cancel» возвращается значение *null*.

Синтаксис:

*prompt("вопрос/запрос","значение по умолчанию");*

Ниже приводится код веб-страницы, в которой пользователь имеет возможность выбрать цвет текста с помощью диалогового элемента

Пример 6

|  |
| --- |
| <html>  <body>  // *здесь будет отображаться текст*  <div id="c" style="color:blue">Вы выбрали цвет текста: черный</div>  <script language="JavaScript">  // *пользователь выбирает цвет текста*  var tcolor = prompt("Выберите цвет текста: red, blue, green, yellow, black","black");  // *задается текст*  document.getElementById("c").innerHTML = "Вы выбрали цвет текста: " + tcolor;  // *задается цвет текста*  document.getElementById("c").style.color = tcolor;  </script>  </body>  </html> |

**Порядок выполнения работы.**

Подготовьте на основе примеров 1-2, 4-6 соответствующие веб-страницы и просмотрите их с помощью браузера.

**Контрольное задание.** Создание таблицы случайно выбранных цветов.

Взяв за основу сценарий построения таблицы умножения, постройте таблицу случайно выбранных цветов. Цвет ячейки таблицы задается с помощью атрибута *bgcolor*. Цвет ячейки описывается в рамках трехкомпонентной модели RGB, например:

*<td bgcolor=”#cb043a”>*.

Для генерации каждой компоненты можно использовать генератор случайных чисел с помощью методов объекта *Math* и преобразование в шестнадцатиричный формат:

*color = Math.round(255.0\*Math.random());*

*r = color.toString(16);*

Результирующий цвет образуется путем конкатенации компонентов:

*color = r + g + b;*

Примерный вид результата работы сценария:

