Лабораторная работа

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 selected 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>
 <HEAD>
  <TITLE>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 C++. 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 + c = "); document.write(a+c);
    document.write("<BR>");
    document.write("c + a = "); document.write (c + a);
    document.write ("<BR>");
    document.write ("a + b + c = "); 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 рис. 8.1.

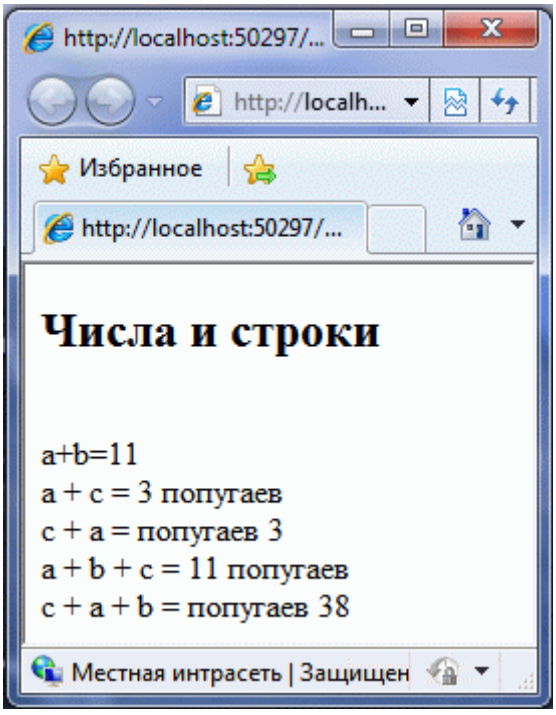


Рис. 8.1. Скриншот веб-страницы с JavaScript-сценарием

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 C++, в основном применимы и в сценариях JavaScript.

В JavaScript дополнительно определены языковые конструкции, отсутствующие в C++, 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> </td>");
        for (i = 1; i < 10; i++)
          document.write("<td>"+i+" </td>");
        document.write("</tr>");
        for (i = 1; i < 10; i++)
        {
          document.write("<tr><td>" + i + " </td>");
          for (j = 1; j < 10; j++)
          {
            document.write("<td  bgcolor='#00ffa0'>" + (i*j)
                      + " </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 *C* для идентификации функции используется имя, при вызове функции могут передаваться параметры, а по окончании выполнения возвращаться значение. Однако, в отличие от *C*, 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 *C*.
- 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.

2. Диалоговые элементы

В JavaScript поддерживается работа со следующими диалоговыми элементами интерфейса:

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

   Синтаксис:

   ```
   alert("сообщение");
   ```

2. Confirm. Применяется для выбора пользователем одного из двух вариантов ответа "Да/Нет". Соответственно Confirm возвращает значение true/false.

   Синтаксис:

   ```
   confirm("вопрос");
   ```

3. 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 2

```
<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>
```

3. Обработка событий в JavaScript

Популярность JavaScript во многом обусловлена именно тем, что написанный на нем сценарий может реагировать на действия пользователя и другие внешние события. Каждое из событий связано с тем или иным объектом: формой, гипертекстовой ссылкой или даже с окном, содержащим текущий документ.

В качестве примеров внешних событий, на которые могут реагировать объекты JavaScript, можно привести следующие.

1. окончание загрузки документа в окно (или окончание загрузки документов во все фреймы окна). Это событие связано с объектом window;
2. щелчок мышью на объекте. Это событие может быть связано с интерактивным элементом формы или с гипертекстовой ссылкой;
3. получение объектом фокуса ввода. Это событие может быть связано с объектами типа Text, Password и с другими интерактивными элементами;
4. передача на сервер данных, введенных пользователем с помощью интерактивных элементов. Связывается с формой.

Обработка события производится с помощью специально предназначенного для этого фрагмента кода, называемого *обработчиком события*. Для каждого события JavaScript предоставляет свой обработчик. Однако при построении сценария можно создавать собственный обработчик события и использовать его вместо обработчика, заданного по умолчанию.

Имя обработчика определяет, какое событие он должен обрабатывать. Так, для того чтобы сценарий нужным образом отреагировал на щелчок мышью, используется обработчик с именем onClick, для обработки события, заключающегося в получении фокуса ввода, - обработчик onFocus.

Для того чтобы указать интерпретатору JavaScript на то, что обработкой события должен заниматься обработчик, необходимо включить в HTML-дескриптор следующее выражение:

```
имя_обработчика="команды_обработчика"
```

Это выражение включается в тэг, описывающий объект, с которым связано событие.

Например, если необходимо обработать событие, заключающееся в получении фокуса полем ввода, дескриптор, описывающий этот интерактивный элемент, должен иметь примерно следующий вид:

```
<input type="text" name="Inform" onFocus="handleFocus();">
```

Имя обработчика является одним из атрибутов HTML-дескриптора, а команды, предназначенные для обработки события, выступают в роли значения этого атрибута. В данном случае обработка события производится в теле функции handleFocus(). В принципе, обработчиком может быть не только функция, но и любая последовательность команд JavaScript в виде составного оператора.

Следующий пример демонстрирует обработку события, связанного с наведением курсора мыши на гиперссылку:

```
<a href = "http://www.myhp.edu"
    onmouseover="alert('An onMouseOver event'); return false">
<img src="my.gif" width="50" height="50">
</a>
```

Ниже приводится полный текст HTML документа с JavaScript сценарием, в котором обрабатывается событие нажатия кнопки мыши в области содержимого документа, и определяется, какая именно из них была нажата:

Пример 3.

```
<html>
  <head>
    <script language = "javascript">
      function whichButton(event)
      {
      if (event.button == 2)
        {
        alert("Вы щелкнули правой кнопкой мыши!");
        }
      else
        {
        alert("Вы щелкнули левой кнопкой мыши!");
        }
      }
    </script>
  </head>
```

```
  <body onmousedown="whichButton(event)">
    <p>Щелкните любой кнопкой мыши в любом месте документа</p>
  </body>

</html>
```

Таким образом, для того чтобы обработать какое-либо стандартное событие в браузере, необходимо добавить в подходящий HTML тэг атрибут, соответствующий этому событию, указав в качестве значения атрибута имя JavaScript функции. Список атрибутов, которые определены для HTML тэгов приводится ниже:

IE: Internet Explorer, F: Firefox, O: Opera, W**3**C: *стандарт*

Атрибут	**Описание**	**Номер версии браузера**			**W3C**
		IE	F	O	
onabort	Прерванная загрузка изображения	4	1	9	Да
onblur	утрата фокуса элементом	3	1	9	Да
onchange	Изменение содержимого в поле ввода	3	1	9	Да
onclick	Щелчок мыши на объекте	3	1	9	Да
ondblclick	Двойной щелчок мыши на объекте	4	1	9	Да
onerror	Ошибка при загрузке изображения или документа	4	1	9	Да
onfocus	Получение фокуса элементом	3	1	9	Да
onkeydown	Нажатие клавиши	3	1	Нет	Да
onkeypress	Клавиша нажата	3	1	9	Да

onkeyup	Отжатие клавиши	3	1	9	Да
onload	Завершение загрузки страницы или изображения	3	1	9	Да
onmousedown	Нажатие кнопки мыши	4	1	9	Да
onmousemove	Перемещение курсора мыши	3	1	9	Да
onmouseout	Смещение курсора мыши с объекта	4	1	9	Да
onmouseover	Наведение курсора мыши на объект	3	1	9	Да
onmouseup	Отжатие кнопки мыши	4	1	9	Да
onreset	Кнопка "Reset" нажата	4	1	9	Да
onresize	Изменение размера окна	4	1	9	Да
onselect	Выделение текста	3	1	9	Да
onsubmit	Кнопка "Submit" нажата	3	1	9	Да
onunload	Уход с веб-страницы	3	1	9	Да

4. Использование регулярных выражений в JavaScript

При поиске по тексту можно использовать шаблон, описывающий подстроку. В JavaScript такой шаблон может быть описан с помощью объекта RegExp. В простейшем случае такой шаблон описывает отдельный символ, однако имеет смысл его использовать для регулярных выражений.

Следующий ниже код описывает RegExp объект с именем pttn, содержащий регулярное выражение, описывающее целое десятичное число:

```
var pattn = new RegExp("/[0-9]+/");
```

Объект RegExp имеет 3 встроенных метода: test(), exec() и compile().

- Метод test() выполняет поиск по шаблону:

- ```
  var pattn = new RegExp("[0-9]+");
  ```
- ```
  document.write(pattn.test("38 попугаев"));
  ```

  Результат:

  ```
  true
  ```
- Метод exec() выполняет поиск подстроки по шаблону и возвращает найденные соответствия; если соответствий нет, возвращается значение null:
- ```
  var pattn=new RegExp("[0-9]+");
  ```
- ```
  document.write(pattn.exec("38 попугаев"));
  ```

  Результат:

  ```
  38
  ```

  Если необходимо найти все соответствия, то при вызове конструктора RegExp следует указать дополнительный параметр "g", указывающий на необходимость глобального поиска.

  Пример 8

  ```
  var pattn = new RegExp("[0-9]+", "g");
  do
  {
    result  = pattn.exec("1 попугай, 2 попугая,…, 38 попугаев");
    document.write(" " + result);
  }
  while (result != null)
  ```

  Результат:

  ```
  1 2 38 null
  ```
- Метод compile() применяется для изменения ранее созданного шаблона.

  Пример 4

  ```
  var pattn = new RegExp("[0-5]+");
  document.write(pattn.exec("38 попугаев"));

  pattn.compile("[6-9]+");
  document.write(";" + pattn.exec("38 попугаев"));
  ```

  Результат:

  ```
  3;8
  ```

Порядок выполнения работы.

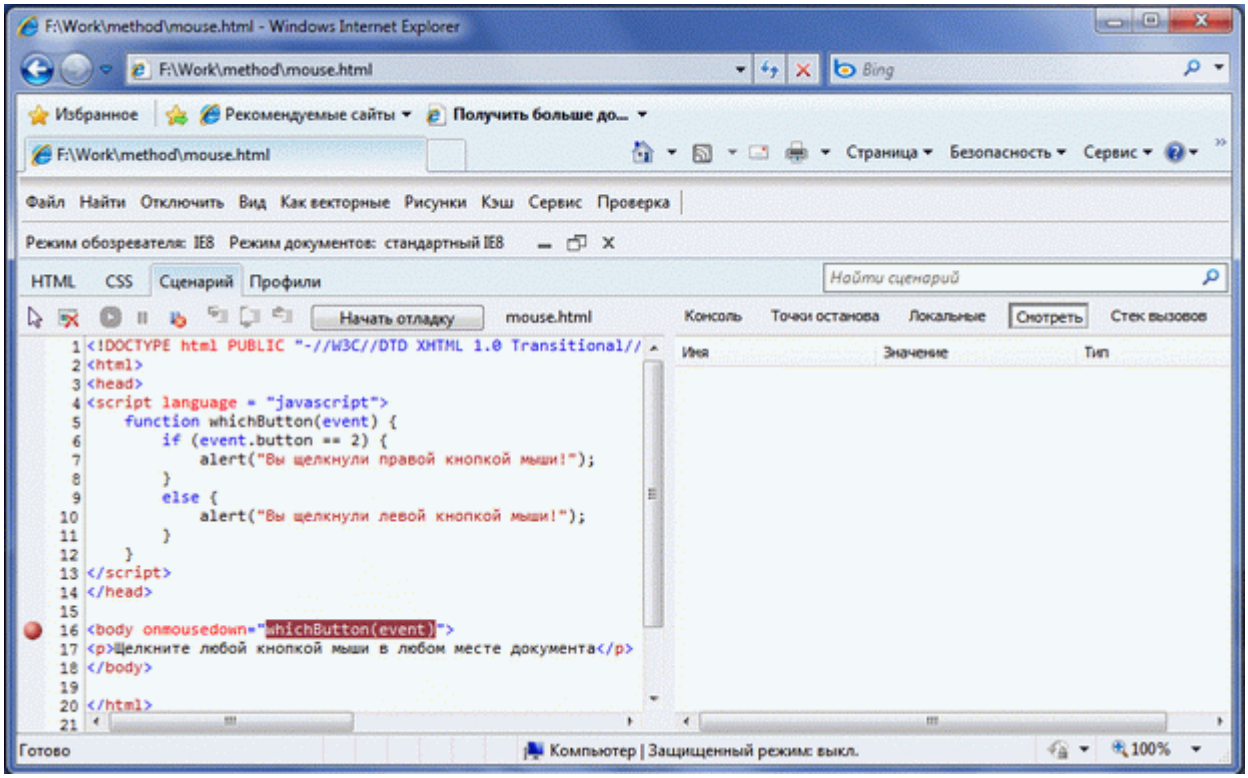
Задание 1. Выполнение сценариев из примеров.

Используя Microsoft WebMatrix, подготовьте на основе примеров 1-4 соответствующие веб-страницы и просмотрите их с помощью веб-браузера IE9.

Задание 2. Отладка JavaScript сценариев с помощью инструментов разработчика в IE9.

Внимание: для отладки JavaScript сценариев в веб-браузере необходимо выбрать соответствующую опцию (Свойства обозревателя > Дополнительно > Обзор > Отключить отладку скриптов).

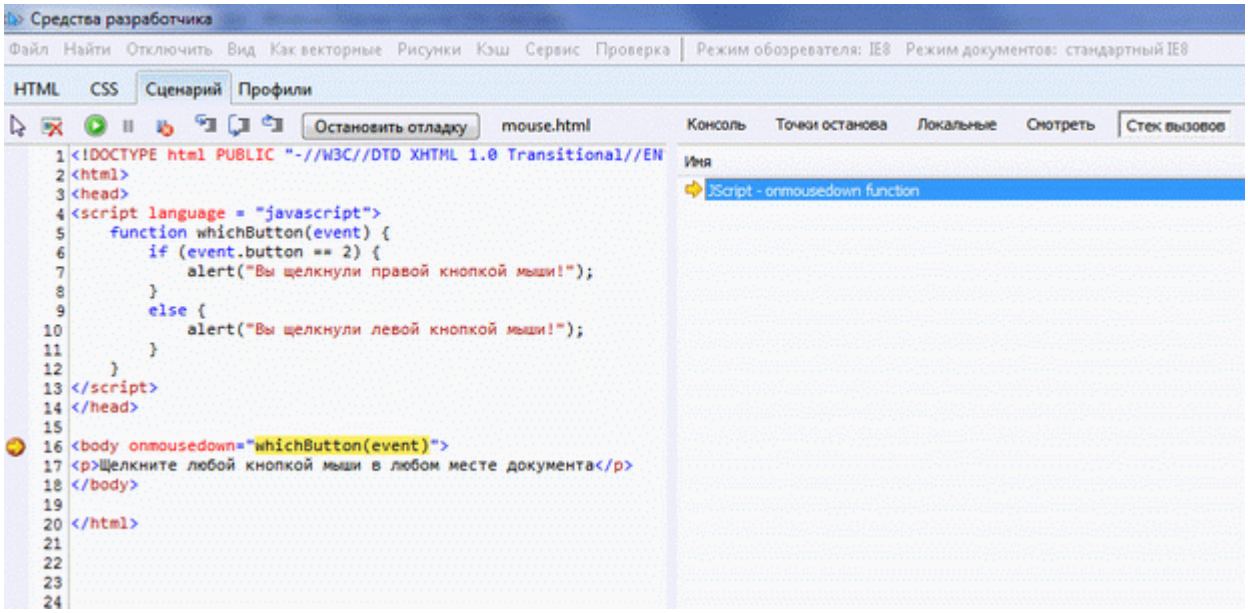
1. Используя код из примера 3 создайте в WebMatrix веб-страницу mouse.html и загрузите ее в веб-браузере IE9.
2. Вызовите панель инструментов разработчика (<F12>).
3. Выберите закладку "Сценарий", затем установите точку останова в строке 16 (либо щелкните на номере строки, либо установить с помощью контекстного меню).



увеличить изображение

Рис. 8.2. Окно отладчика JavaScript-сценариев

4. Нажмите кнопку "Начать отладку".
5. Закройте окно инструментов разработчика и обновите веб-страницу в браузере (<F5>).
6. Щелкните любой кнопкой мышки на тексте в окне браузера. При этом должно появится следующее окно отладчика.

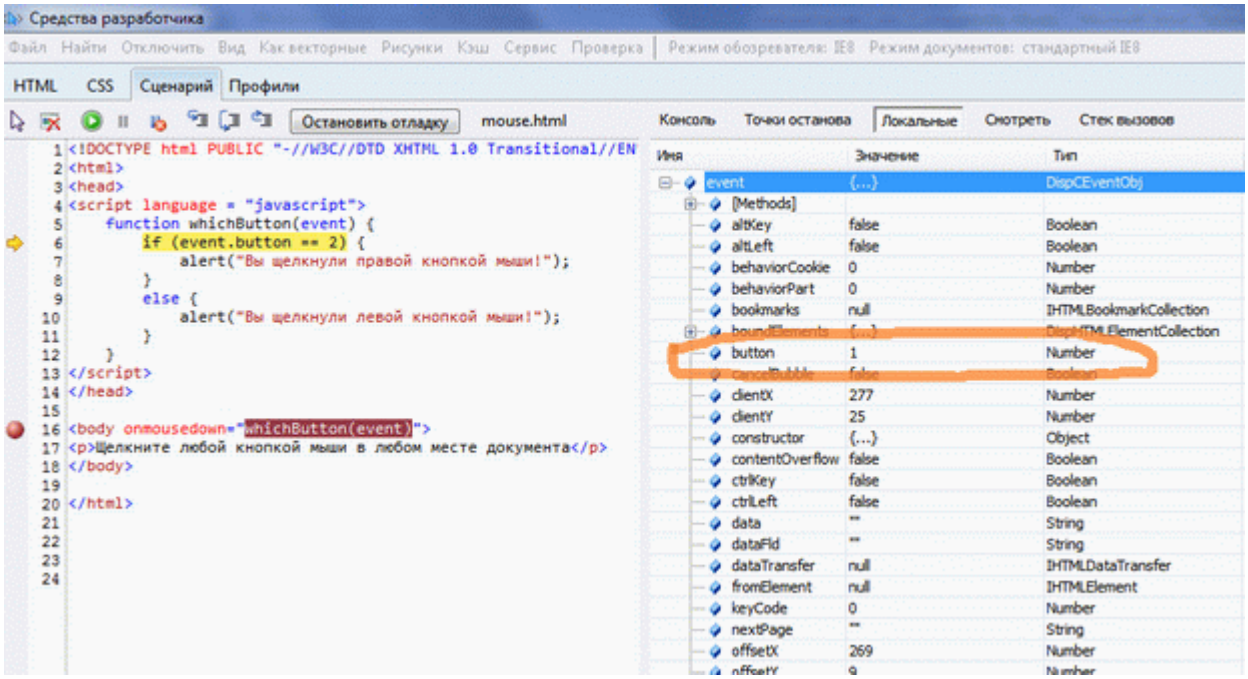


увеличить изображение

Рис. 8.3. Начало отладчики JavaScript-сценария

Отладчик автоматически остановится на заданной ранее точке останова.

7. В правом окне выберите просмотр "С*тека вызовов*". При нажатии клавиши <F11> отладчик продолжит работу уже внутри функции-обработчика клика мыши.
8. Выберите просмотр "*Локальные*". Раскрыв дерево объекта Event, можно будет увидеть значение его атрибута button.



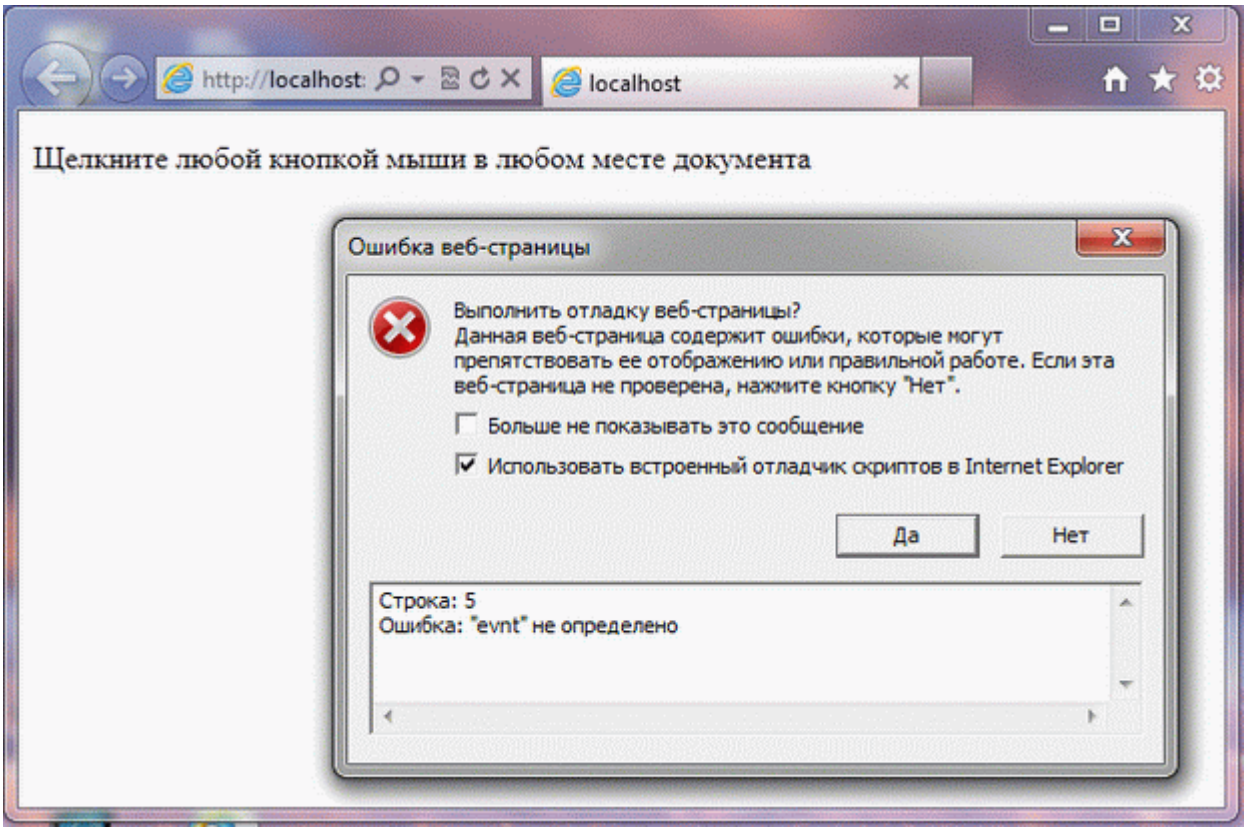
увеличить изображение

Рис. 8.4. Просмотр значений локальных переменных и объектов в отладчике JavaScript-сценария

9. Продолжите пошаговое выполнение сценария до конца.

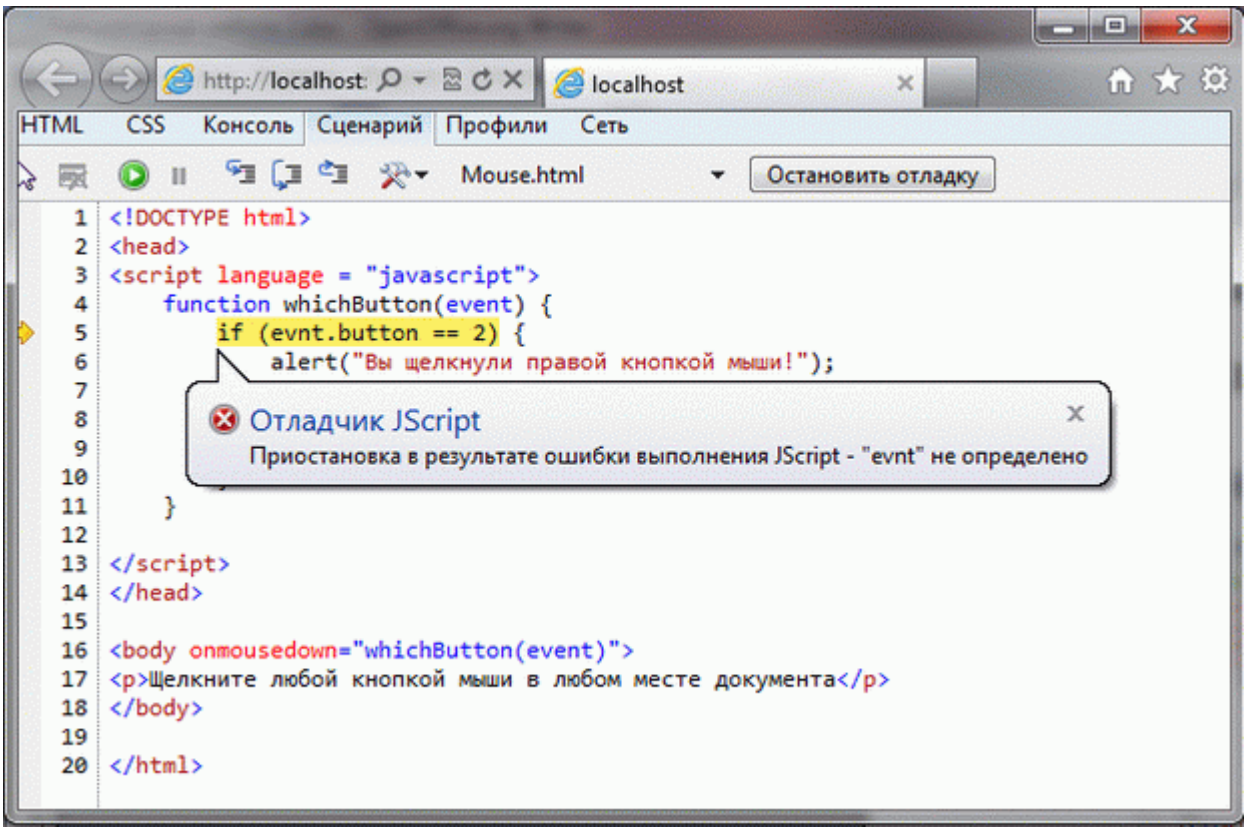
При поиске и исправлении ошибок в JavaScript сценариях удобно использовать функцию автоматической приостановки отладчика сценария в строке, где возникает ошибки. Для этого перед запуском сценария необходимо нажать кнопку  .

- Измените в 5 строке имя объекта event на evnt и сохраните веб-страницу.
- Загрузите страницу в браузере и вызовите окно инструментов разработчика.
- Нажмите кнопку кнопку "Начать отладку".
- Закройте окно отладчика, обновите страницу в браузере и щелкните кнопкой мышке на тексте документа.



увеличить изображение

Рис. 8.5.1. Сообщение об ошибке



увеличить изображение

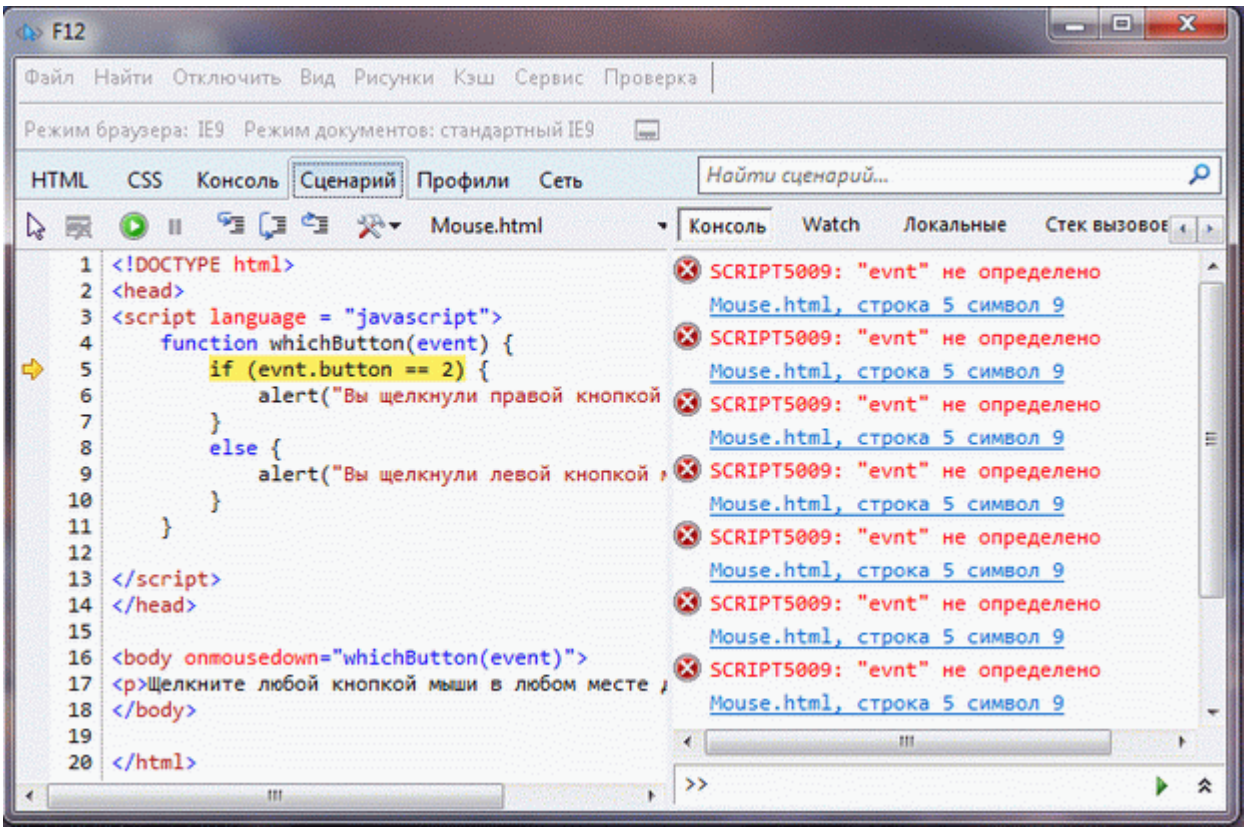
Рис. 8.5.2. Автоматическая приостановка отладчика JavaScript-сценария при ошибке

В результате произойдет ошибка появится сообщение об ошибке, как показано на рис. 8.5.1. Нажмите "Да", после этого вы увидите окно отладчика (как на рис. 8.5.2).

1. Переключитесь в режим "Консоль" и введите в текстовое поле внизу команду:

```
2. console.log("Event.button = " + event.button)
```

3. Нажмите кнопку "Запускать сценарий" внизу от поля ввода команды.



увеличить изображение

Рис. 8.6. Выполнение команд отладки в консольном режиме отладчика JavaScript-сценария

В результате вы увидите значение Event.button в точке останова.

Консоль используется для записи в журнал событий вместо использования диалога window.alert(). Для разных типов сообщений можно использовать разные журналы, например:

- console.log
- console.info
- console.warn
- console.error
- console.assert

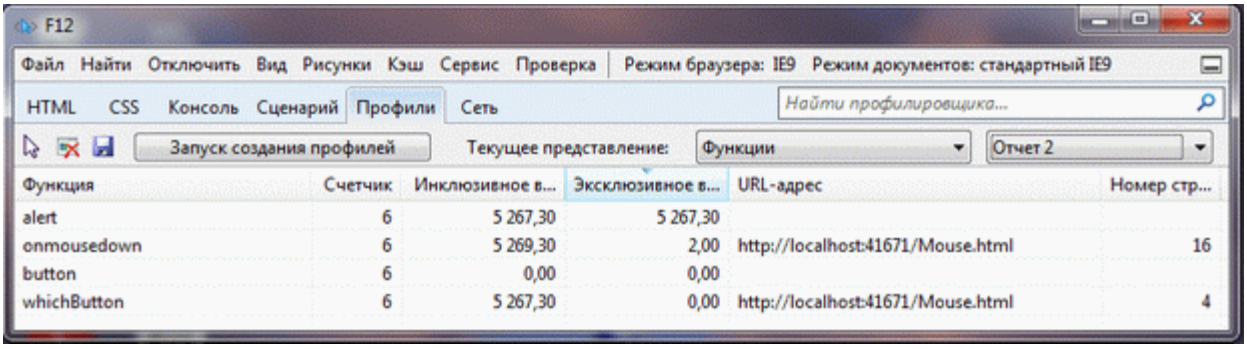
С помощью контекстного меню консоли (правая кнопка мыши) можно очистить все сообщения или применить фильтр для отбора сообщений.

Задание 3. Отладка JavaScript сценариев с помощью инструментов разработчика в IE9.

Профилировщик JavaScript служит для повышения производительности сайта. Он предоставляет данные о времени, затраченном каждым из методов JavaScript сайта и встроенными функциями JavaScript, такими как сцепление строк. Профилировщик JavaScript можно запустить и остановить в любое время при выполнении приложения, поэтому можно собрать множество данных для нужных сценариев.

- Загрузите созданную ранее веб-страницу mouse.html в браузере IE9.
- Вызовите окно инструментов разработчика.
- Выберите закладку "Профили" и нажмите кнопку "Запуск создания профилей".
- Закройте окно инструментов разработчика и выполните несколько действий с мышью в окне веб-браузера, как предусмотрено в JavaScript сценарии.
- Вызовите снова окно инструментов разработчика.

- Нажмите кнопку "Остановка создания профилей"



увеличить изображение

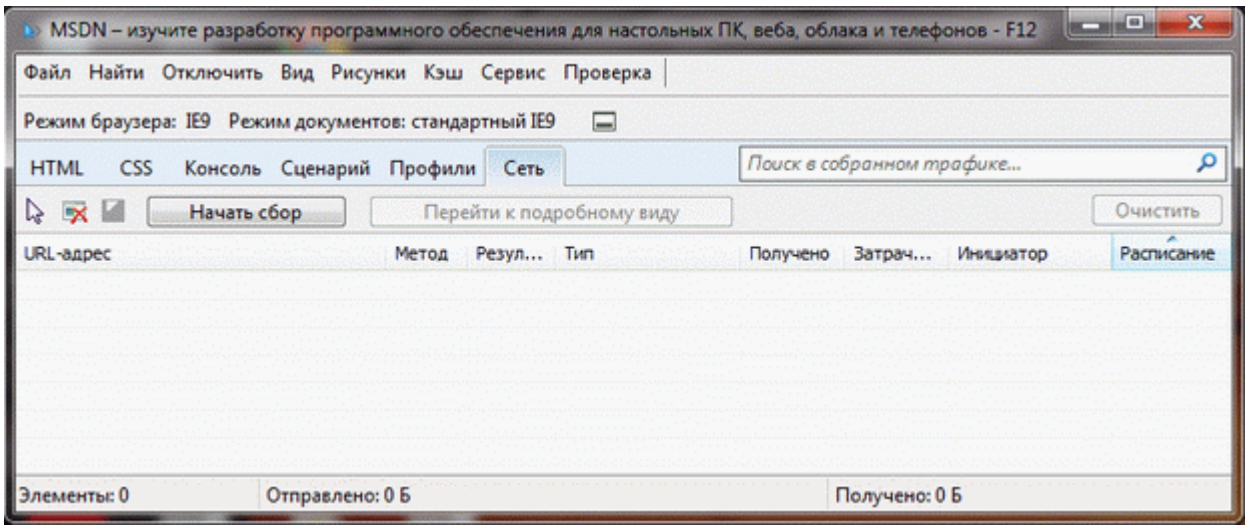
Рис. 8.7. Окно профилировщика JavaScript-сценариев

По умолчанию данные отображаются в представлении функций, где перечисляются все использованные функции. В отчете профилировщика содержится URL-адрес и номер строки функции, чтобы было проще найти этот код в приложении. Можно сохранить данные профиля в формате CSV.

Задание 4. Отслеживание сетевого трафика с помощью инструментов разработчика в IE9.

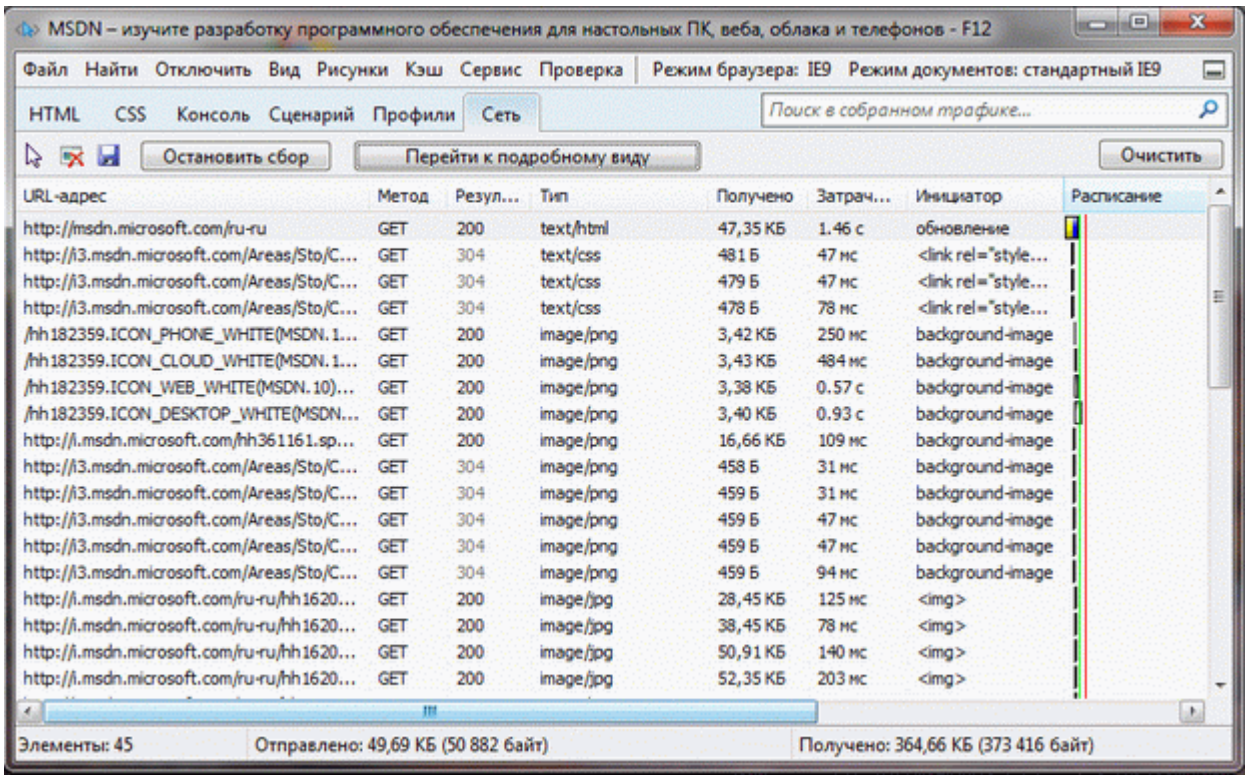
Вкладка "Сеть" на верхней панели окна инструментов разработчика (F12) позволяет увидеть разработчику, какие ресурсы использует веб-страница, включая данные, которые отправлены клиентом и получены от сервера. Разработчики могут использовать эту информацию для ослеживания ошибок в ответах веб-сервера, а также для отладки AJAX-запросов, поскольку может возникнуть необходимость изучения данные, как отправленных, так и полученных от веб-сервера.

1. Чтобы найти вкладку "Сеть", запустите инструменты разработчика (F12) в IE9. Выберите вкладку "Сеть".



   увеличить изображение

   Рис. 8.8. Окно отслеживания сетевого трафика
2. Нажмите "Начать сбор", чтобы начать запись сетевого трафика. Данный инструмент не будет захватывать данных, пока вы не нажмете эту кнопку, потому что сбор сетевых данных влияет на производительность и потребление памяти.
3. Обновите страницу в окне веб-браузера для того, чтобы увидеть зарегистрированные сетевые запросы в сводном виде в динамике.



увеличить изображение

Рис. 8.9. Сбор сетевого трафика

Сводном виде отображается список запросов веб-страницы странице.

Запросы могут исходить от:

- HTML и CSS документов
- JavaScript сценариев
- Динамическая настройка src-атрибутов
- Объектов XmlHttpRequest и XDomainRequest
- Расширений веб-браузера
- Элементов управления ActiveX типа Flash
- Панели BHO и Explorer

Для каждого запроса можно перейти к детальному представлению либо по двойному щелчку левой кнопки мыши, либо нажав кнопку "перейти к подробному виду".

4. Остановите захват, нажав кнопку "Остановить сбор".
5. Сохраните результаты сбора, нажав кнопку сохранения на панели. При этом можно выбрать один из двух форматов: XML или CSV.

Контрольные задания.

Задание 1. 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="#c0a145">. 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:

увеличить изображение

Задание 2.

С помощью отладчика сценариев в наборе инструментов разработчика веб-браузера IE9 проследите как исполняется созданный вами в первом контрольном задании JavaScript сценарий в пошаговом режиме. Проследите также, как изменяются значения локальных переменных. Можете использовать отладчик для поиска и устранения ошибок в процессе разработки сценария в предыдущем контрольном задании.

Задание 3.

С помощью профилировщика в наборе инструментов разработчика веб-браузера IE9 соберите статистику по функциям, используемым в сценарии, созданном в контрольном задании 1.