JavaScript

Краткое введение в JavaScript

JavaScript это:

- Интерпретируемый язык. Его интерпретатор обычно встроен в браузер.
- Основное назначение – определять «динамическое» поведение страниц при загрузке (формирование страницы перед ее открытием) и при работе пользователя со страницей (UI элементы).
- Текст на JavaScript может быть вложен в HTML-страницу непосредственно или находиться в отдельном файле (как CSS).
- Похож на языки Java и C# синтаксически, но сильно отличается от них по внутреннему содержанию.

Характеристика JavaScript

- Язык объектно-ориентированного программирования. Объекты в языке имеют «тип», «атрибуты» и «методы»

```
"John,Jane,Paul,Michael".split(",").length
```

- Переменные не имеют заранее заданного типа, то есть в разные моменты времени могут содержать значения разных типов

```
var number = 25;  number = (number < 0);  number = "25";
```

- Типы объектов могут быть: number, string, function, object, undefined. Оператор typeof позволяет «вычислить» тип объекта.

```
typeof 25 == "number"   typeof null == "object"
```

Область применения

- Веб-приложения
  - AJAX
  - Comet
  - Браузерные операционные системы
- Букмарклеты
- Пользовательские скрипты в браузере
- Серверные приложения
- Мобильные приложения
- Виджеты
- Прикладное программное обеспечение
- Манипуляция объектами приложений
- Офисные приложения
- Microsoft Office
- OpenOffice.org
- Обучение информатике

Включение JavaScript в HTML-страницу

- Внутри страницы.
- Внутри тега.
- Отдельно от разметки.
- В отдельном файле.

JavaScript внутри страницы

- Для добавления JavaScript-кода на страницу, можно использовать теги **`<script></script>`.**
- Скрипт, выводящий модальное окно с классической надписью «Hello, World!» внутри браузера:

```
<script type="text/javascript">

    alert('Hello, World!');

</script>
```

JavaScript внутри тега

- Спецификация HTML описывает набор атрибутов, используемых для задания обработчиков событий.

Пример использования:

```
<a href="delete.php"

onclick="return confirm('Вы уверены?');">

Удалить</a>
```

JavaScript отдельно от разметки

- Использование кода JavaScript в контексте разметки страницы в рамках ненавязчивого JavaScript расценивается как плохая практика.

```
<a href="delete.php" id="alertLink">Удалить</a>

window.onload = function() {
    var linkWithAlert = document.getElementById("alertLink");
    linkWithAlert.onclick = function() {
        return confirm('Вы уверены?');
    };
};
```

JavaScript в отдельном файле

```
<script type="text/javascript"
src="http://Путь_к_javascript_файлу"></script>
```

Тег script имеет несколько атрибутов:

- **type**
- **src**
- **charset**
- **defer**
- **language**

Два простых примера

Метод document.write используется для непосредственного включения HTML-текста в содержимое страницы, например, можно сгенерировать длинный текст в параграфе:

```
<body>
  <p>
    <script type="text/javascript">
      for (var i = 0; i < 100; ++i) {
        document.write("Hello, world! ");
      }
    </script>
  </p>
</body>
```

Два простых примера (продолжение)

Во втором примере датчик случайных чисел используется для генерации случайной ссылки (из заданного набора):

```
<body>
  <p>
    <script type="text/javascript">
      var rand = Math.random();          // в диапазоне: [0, 1)
      var numb = Math.floor(rand * 10);
      var image = "images/image" + numb + ".jpg";
      var insert = "<img class=\"floatRight\" src=\"" +
                   image + "\" alt=\"Фотография цветочка\"/>";
      document.write(insert);
    </script>
  </p>
</body>
```

Основные встроенные типы

Есть набор встроенных «классов», порождающих «объекты», различающиеся набором атрибутов и методов. Программисты могут динамически изменять поведение этих «классов» и создавать свои собственные. Каждый «класс» является объектом, у которого есть «прототип», определяющий набор атрибутов и методов у всех вновь создаваемых объектов этого класса.

Типы, встроенные в язык, это:

- Number : 64-х-разрядные числа с плавающей точкой.
- String : строки с символами в формате Unicode.
- Array : массивы с переменными границами.
- Function : Функции. Каждая функция, кроме того, может служить конструктором объекта.
- Boolean, Date, Math, RegExp : логические значения, даты,…

Объекты, встроенные в браузеры

Основные из них это:

- **window** : представляет «глобальный контекст» и позволяет работать с атрибутами и методами окна.
- **document** : загруженная страница со своей структурой элементов.
- **navigator** : объект, представляющий браузер и его свойства.
- **location** : характеристики текущего URL (порт, хост и т.п.).
- объекты, представляющие элементы различных типов в HTML-странице, такие как <body>, <link>, <img> и т.п.
- события (**events**), возникающие от действий пользователя, например, нажатие кнопки мыши (click), загрузка новой страницы (load) и т.д.

События и реакции на них

Имеется большое количество событий, которые можно разделить на следующие классы:

- события от мыши (click, dblclick, mousedown,…);
- события от клавиатуры (keypress, keydown,…);
- события от элементов ввода (focus, submit, select,…);
- события страницы (load, unload, error,…);

Один из способов программирования состоит в определении реакции на события непосредственно в описании элемента, например:

```
<p>День независимости России <span style="color: blue;
text-decoration: underline;" onclick= "alert('Осталось ' +
DaysToDate(12, 6) + 'дней');"> 12 июня</span>.
</p>
```

Недостаток этого способа: javascript-текст опять смешивается с содержанием страницы.

Работа с таймером

Можно создать таймер и определить реакцию на событие от таймера.

```
var timer = setTimeout(func, timeinterval);
```

func – это функция или строка с кодом; timeinterval – время в миллисекундах. Таймер срабатывает один раз и запускает функцию.

```
function launchTimer() { setTimeout("alert('Зенит – чемпион!');", 2000); }
```

Теперь можно запустить этот таймер, например, по событию click:

```
<body><p>Нажми <span onclick="launchTimer()
;">сюда!</span></p> </body>
```

Пока событие еще не случилось, таймер можно остановить:

```
var timer = setTimeout(func, timeinterval);
clearTimeout(timer);
```

Работа с интервальным таймером

Таймер может срабатывать многократно через равные промежутки времени. Такой таймер создается с помощью функции setInterval и останавливается с помощью функции clearInterval.

```
var timer = setInterval(func, timeinterval);
function launchInterval() {
    timer = setInterval("alert('Зенит – чемпион!');", 2000);
}
function stopTimer() {
    if (timer) clearInterval(timer);
    timer = null;
}

<body>
    <p>Нажми <span onclick="launchInterval();">сюда,</span>
чтобы запустить.</p>
    <p>Нажми <span onclick="stopTimer();">сюда,</span> чтобы
остановить.</p> </body>
```

Объектная модель браузера

- Объектная модель браузера – браузероспецифичная часть языка, являющаяся прослойкой между ядром и объектной моделью документа.
- Основное предназначение объектной модели браузера – управление окнами браузера и обеспечение их взаимодействия.
- Каждое из окон браузера представляется объектом window, центральным объектом BOM.
- Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке WHATWG и W3C.

Объектная модель браузера

- Управление фреймами.
- Поддержка задержки в исполнении кода и зацикливания с задержкой.
- Системные диалоги.
- Управление адресом открытой страницы.
- Управление информацией о браузере.
- Управление информацией о параметрах монитора.
- Ограниченное управление историей просмотра страниц.
- Поддержка работы с HTTP cookie.

Объектная модель документа

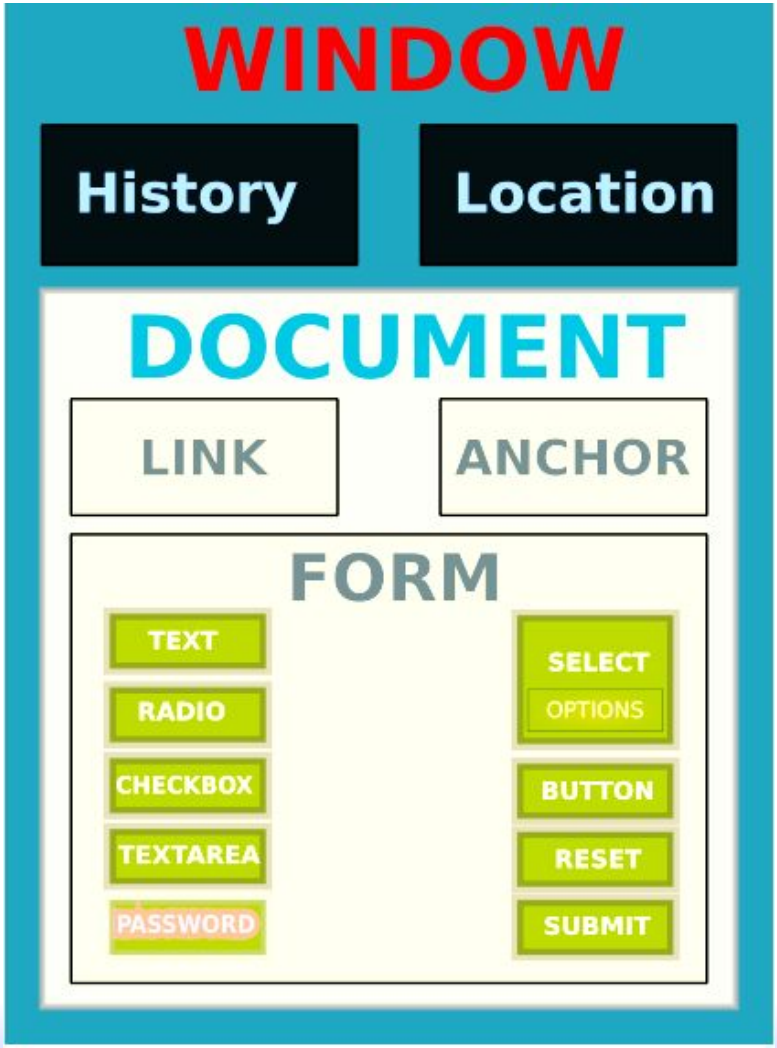
DOM (от англ. Document Object Model – «объектная модель документа») – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Объектная модель документа

- Объектная модель документа – интерфейс программирования приложений для HTML и XML-документов.
- Согласно DOM документу можно поставить в соответствие дерево объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции:
  - получение узлов,
  - изменение узлов,
  - изменение связей между узлами,
  - удаление узлов.

Иерархия объектов в HTML DOM

- Модель DOM не накладывает ограничений на структуру документа.
- Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект.
- Узлы связаны между собой отношениями родительский-дочерний.



jQuery

- jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML.
- Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими.
- Также библиотека jQuery предоставляет удобный API по работе с Ajax.

Возможности

- Кросс-браузерный выбор DOM обьектов
- Перемещение и модификация DOM
- События (Events)
- Работа с CSS
- Визуальные эффекты и анимация
- AJAX-дополнения
- Расширяемость плагинами включая jQuery UI

Использование

Query, как правило, включается в веб-страницу как один внешний JavaScript-файл:

```
<head>

<script type="text/javascript" src="
путь/к/jQuery.js"></script>

</head>
```

Вся работа с jQuery ведѐтся с помощью функции $. Если $ уже использоваться для своих нужд, то можно использовать еѐ синоним — jQuery

Использование

Работу с jQuery можно разделить на 2 типа:

• Получение jQuery-объекта с помощью функции $ и далее работать с ними с помощью различных методов jQuery-объекта.

• Вызов глобальных методов у объекта $, например, удобных итераторов по массиву.

```
$("div.test").addClass("red").slideDown("slow");
```

Плюсы от использования

- Не надо самому писать тривиальный код
- Простота использования и изучения
- Кросс-браузерность
- Отказ от длиных конструкций JS, вроде document.getElementById(”select_list”)
- Библиотека Ajax
- Огромное количество плагинов
- Повсеместное использование а значит кэширование браузерами и ускорение загрузки

jQuery UI

Взаимодействия:

- Пертаскивание (Draggable)
- Вложение (Droppable)
- Изменение размера (Resizable)
- Выделение (Selectable)
- Сортировка (Sortable)

Виджеты:

Аккордион, Автодополнение, Кнопки, Выбор даты, Диалоговые окна, Progressbar, Slider, Вкладки

Примеры

Выбор элемента по классу или идентификатору

```
$("#myDivID")

$(".myDivClass")
```

Если объект с заданным Id существует то сделать его видимым

```
if( $("#myDiv").length )
    $("#myDiv").show();
```

Примеры

Анимация сдвига вправо

```
$("#myDivID:visible").animate({left: ”+=200px” }, ‘slow’);
```

Получение внутреннего html кода обьекта

```
$("#myDiv").html();
```

AJAX

- Асинхронные вызовы: результат приходит не сразу после выполнения вызова, а по готовности ответа
- Callback
- Кэширование
- XmlHttpRequest

XmlHttpRequest

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'phones.json', false);

xhr.send();

if (xhr.status != 200) {
        alert( xhr.status + ': ' + xhr.statusText );
} else {
       alert( xhr.responseText ); // responseText -- текст ответа.
}
```

XmlHttpRequest

```
var xhr = new XMLHttpRequest();

xhr.open('GET', 'phones.json', true);

xhr.send();

xhr.onreadystatechange = function() { // (3)
        if (xhr.readyState != 4) return;

        button.innerHTML = 'Готово!';

       if (xhr.status != 200) {
                alert(xhr.status + ': ' + xhr.statusText);
        } else {
                alert(xhr.responseText);
        }
}
```

Примеры

Работа с AJAX

```
$.ajax({
    url: ”myPage.html”,
    success: function(response) {
        alert(response);
    }
    error: function(xhr) {
        alert(”Error! Status: ” + xhr.status);
    }
});
```

Примеры

```
$.ajax({
        type: "POST",
        url: url,
        data: data,
        success: success,
        dataType: dataType
});
```

События ajax

- **beforeSend**

  Происходит перед выполнением запроса, что позволяет произвести необходимые настройки

- **complete**

  Происходит при завершении запроса (неважно, удачном оно было или нет).

- **error**

  Происходит в случае неудачного завершения запроса.

- **success**

  Происходит в момент удачного завершения запроса.

Callbacks

```
post("http://localhost:8080/api/1.0/fio", postData, function (data)
{

    alert(data);

});
```

```
function post(url, postData, callBack) {

    …

}
```

Ссылки

https://habr.com/post/306716/

https://habr.com/post/341626/

https://habr.com/post/312022/