Восьмая лекция
java for web
JS & Ajax

Введение в JavaScript

JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда был очень популярен язык Java, и маркетологи решили, что схожее название сделает новый язык более популярным.

Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией и к Java не имеет никакого отношения.

Во все основные браузеры встроен интерпретатор JavaScript, именно поэтому они могут выполнять скрипты на странице. Но, разумеется, JavaScript можно использовать не только в браузере.

Что умеет JavaScript?

В браузере JavaScript умеет делать всё, что относится к манипуляции со страницей, взаимодействию с посетителем и, в какой-то мере, с сервером:

Создавать новые HTML-теги, удалять существующие, менять стили элементов, прятать, показывать элементы и т.п.

Реагировать на действия посетителя, обрабатывать клики мыши, перемещения курсора, нажатия на клавиатуру и т.п.

Посылать запросы на сервер и загружать данные без перезагрузки страницы (эта технология называется "AJAX").

Получать и устанавливать cookie, запрашивать данные, выводить сообщения…

…и многое, многое другое!

Привет, мир!

Программы на языке JavaScript можно вставить в любое место HTML при помощи тега SCRIPT. Например:

```
<script>
  alert( 'Привет, Мир!' );
</script>
```

Тег script содержит исполняемый код. Предыдущие стандарты HTML требовали обязательного указания атрибута type, но сейчас он уже не нужен. Достаточно просто <script>.

Браузер, когда видит <script> Начинает отображать страницу, показывает часть документа до script . Встретив тег script, переключается в JavaScript-режим и не показывает, а исполняет его содержимое.

Закончив выполнение, возвращается обратно в HTML-режим и только тогда отображает оставшуюся часть документа.

Попробуйте этот пример в действии, и вы сами всё увидите.

Внешние скрипты, порядок исполнения

Если JavaScript-кода много – его выносят в отдельный файл, который подключается в HTML:

```
<script src="/path/to/script.js"></script>
```

Здесь /path/to/script.js – это абсолютный путь к файлу, содержащему скрипт (из корня сайта). Браузер сам скачает скрипт и выполнит.

Как правило, в HTML пишут только самые простые скрипты, а сложные выносят в отдельный файл.

Браузер скачает его только первый раз и в дальнейшем, при правильной настройке сервера, будет брать из своего кеша.

Благодаря этому один и тот же большой скрипт, содержащий, к примеру, библиотеку функций, может использоваться на разных страницах без полной перезагрузки с сервера.

Асинхронные скрипты: defer/async

Браузер загружает и отображает HTML постепенно. Особенно это заметно при медленном интернет-соединении: браузер не ждёт, пока страница загрузится целиком, а показывает ту часть, которую успел загрузить.

Если браузер видит тег <script>, то он по стандарту обязан сначала выполнить его, а потом показать оставшуюся часть страницы.

Такое поведение называют «синхронным». Как правило, оно вполне нормально, но есть важное следствие.

Если скрипт – внешний, то пока браузер не выполнит его, он не покажет часть страницы под ним.

Атрибут async

Поддерживается всеми браузерами, кроме IE9-. Скрипт выполняется полностью асинхронно. То есть, при обнаружении <script async src="..."> браузер не останавливает обработку страницы, а спокойно работает дальше. Когда скрипт будет загружен – он выполнится.

Атрибут defer

Скрипт также выполняется асинхронно, не заставляет ждать страницу, но есть два отличия от async.

Первое – браузер гарантирует, что относительный порядок скриптов с defer будет сохранён.

Второе отличие – скрипт с defer сработает, когда весь HTML-документ будет обработан браузером.

Структура кода

Раньше мы уже видели пример команды: alert('Привет, мир!') выводит сообщение.

Для того, чтобы добавить в код ещё одну команду – можно поставить её после точки с запятой.

Например, вместо одного вызова alert сделаем два:

```
alert('Привет'); alert('Мир');
```

Как правило, каждая команда пишется на отдельной строке – так код лучше читается:

```
alert('Привет');
alert('Мир');
```

Переменная

Переменная состоит из имени и выделенной области памяти, которая ему соответствует. Для объявления или, другими словами, создания переменной используется ключевое слово **var**:

```
var message;
```

После объявления, можно записать в переменную данные:

Эти данные будут сохранены в соответствующей области памяти и в дальнейшем доступны при обращении по имени:

```
var message;
message = 'Hello!';
alert( message ); // выведет содержимое переменной
```

Для краткости можно совместить объявление переменной и запись данных:

```
var message = 'Hello!';
```

Можно даже объявить несколько переменных сразу:

```
var user = 'John', age = 25, message = 'Hello';
```

Пять примитивных типов данных

var n = 123;

n = 12.345;

Единый тип число используется как для целых, так и для дробных чисел.

Существуют специальные числовые значения Infinity (бесконечность) и NaN (ошибка вычислений). Например, бесконечность Infinity получается при делении на ноль:

alert(1 / 0); // Infinity

Ошибка вычислений NaN будет результатом некорректной математической операции, например:

alert("нечисло" * 2); // NaN, ошибка

Строка «string»

var str = "Мама мыла раму";

str = 'Одинарные кавычки тоже подойдут';

В JavaScript одинарные и двойные кавычки равноправны. Можно использовать или те или другие.

Тип символ не существует, есть только строка.

В некоторых языках программирования есть специальный тип данных для одного символа. Например, в языке Java это char. В JavaScript есть только тип «строка» string. Что, надо сказать, вполне удобно.

Булевый (логический) тип «boolean». У него всего два значения: true (истина) и false (ложь). Как правило, такой тип используется для хранения значения типа да/нет, например:

```
var checked = true;
checked = false;
```

Специальное значение «null». Значение null не относится ни к одному из типов выше, а образует свой отдельный тип, состоящий из единственного значения null:

```
var age = null;
```

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое имеет смысл «ничего» или «значение неизвестно».

Специальное значение «undefined»

Значение undefined, как и null, образует свой собственный тип, состоящий из одного этого значения. Оно имеет смысл «значение не присвоено». Если переменная объявлена, но в неё ничего не записано, то её значение как раз и есть undefined:

```
var x;
alert( x ); // выведет "undefined"
```

Объекты «object»

Объекты в JavaScript сочетают в себе два важных функционала.

Первый – это ассоциативный массив: структура, пригодная для хранения любых данных. В этой главе мы рассмотрим использование объектов именно как массивов.

Второй – языковые возможности для объектно-ориентированного программирования.Он используется для коллекций данных и для объявления более сложных сущностей.

Объявляются объекты при помощи фигурных скобок {...}, например:

```
var user = { name: "Вася" }
```

Основные операции с объектами – это создание, получение и удаление свойств.

Для обращения к свойствам используется запись «через точку», вида объект.свойство, например:

```
var person = {}; // пока пустой
person.name = 'Вася';
person.age = 25;
```

Основные операторы

Для работы с переменными JavaScript поддерживает все стандартные математические операторы Java.

Это обычные сложение +, умножение *, вычитание и так далее.

Операторы сравнения также знакомы нам из Java:

Это больше/меньше: a > b, a < b.

Больше/меньше или равно: a >= b, a <= b.

Равно a == b. Для сравнения используется два символа равенства '='. Один символ a = b означал бы присваивание.

«Не равно» !=.

Взаимодействие с пользователем: prompt, confirm

Функция prompt принимает два аргумента:

result = prompt(title, default);

Она выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL.

Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Esc на клавиатуре.

Вызов prompt возвращает то, что ввёл посетитель – строку или специальное значение null, если ввод отменён.

confirm

result = confirm(question);

confirm выводит окно с вопросом question с двумя кнопками: OK и CANCEL.

Результатом будет true при нажатии OK и false – при CANCEL(Esc).

Например:

```
var isAdmin = confirm("Вы - администратор?");
alert( isAdmin );
```

Функции

Зачастую нам надо повторять одно и то же действие во многих частях программы. Чтобы не повторять один и тот же код во многих местах, придуманы функции. Примеры встроенных функций вы уже видели – это alert(message), prompt(message, default) и confirm(question). Но можно создавать и свои.

Пример объявления функции:

```
function showMessage() {
  alert( 'Привет всем присутствующим!' );
}
```

Вначале идет ключевое слово function, после него имя функции, затем список параметров в скобках (в примере выше он пустой) и тело функции – код, который выполняется при её вызове.

Блоки if/else, switch, for, while, do..while не влияют на область видимости переменных.

Неважно, где именно в функции и сколько раз объявляется переменная. Любое объявление срабатывает один раз и распространяется на всю функцию.

Функцию можно вызвать с любым количеством аргументов.

Функция может возвратить результат, который будет передан в вызвавший её код.

jQuery

jQuery — быстрый и надежный способ программирования на JavaScript и Ajax. Набор функций Ajax, доступных через jQuery, значительно упрощает Ajax-разработку, позволяя писать меньше кода и применять дополнительные методы и обработчики событий, охватывающие большинство ситуаций.

С jQuery требуется минимальное количество кода даже при разработке сложных функций, так что процесс идет намного быстрее. Благодаря этим инструментам Ajax становится более практичным способом разработки Web-сайта или приложения в сжатые сроки.

jQuery - это, по существу, библиотеки JavaScript, которая упрощает разработку JavaScript. Она минимизирует код, который нужно написать, благодаря множеству встроенных возможностей, обычно требующих создания специальных функций или объектов.

Для этого сначала надо скачать саму библиотеку на сайте разработчиков или с нашего сайта, при необходимости разархивировать и перенести ее (библиотеку) в ту же папку, где лежат наши html-страницы (это необязательно, но адреса для всех последующих примеров будут написаны, исходя из такой структуры).

Теперь нам надо подключить jQuery к html-странице.

Синтаксис оператора jQuery

Приблизительно синтаксис оператора jQuery можно представить следующим образом:

$('селектор').действие('свойства действия')

где селектор - элемент или элементы, с которыми мы будем что-либо делать.

действие - что именно мы будем делать с выбранными элементами. Мы можем добавить какой-либо эффект, css-стиль, изменить html-код и т.д.

свойства действия - если они предусмотрены действием.

Пример:

```
$("#lok").css("border", "1px solid red");
$("input[name='Main']").css("color", "red");
```

В jQuery имеется три категории методов: одни манипулируют с элементами, подходящими по шаблону; вторые возвращают значения элемента, а третьи изменяют сами элементы.

html(val) - добавляет html-код в выбранные элементы.

Пример:

```
$("div.sp").html("<span>Привет</span>");
```

val() - возвращает текстовое содержимое элемента / значение элемента.

Пример:

```
$("p").text();
```

Данная инструкция вернет текст из параграфа.

val(val) - установит текст для элемента / значение для элемента.

Пример:

```
$("p").text("Привет!");
```

Данная инструкция в параграфе напишет слово "Привет!".

attr(name) - обеспечивает доступ к значению указанного атрибута первого элемента в наборе.

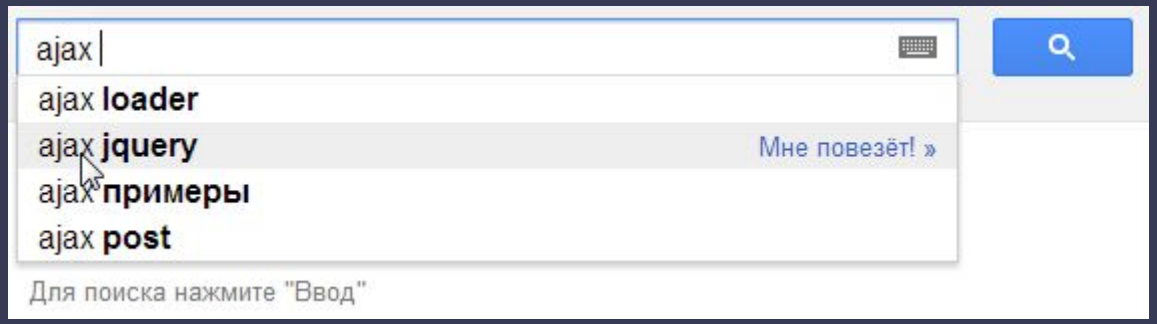
Пример:

```
var a=$("i").attr("title");
$("div").text(a);
```

Что такое Ajax?

Asynchronous Javascript And Xml - технология для взаимодействия с сервером без перезагрузки страниц.За счет этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.

Технология AJAX, как указывает первая буква A в ее названии - асинхронна, т.е браузер, отослав запрос, может делать что угодно, например, показать сообщение **об** ожидании ответа, прокручивать страницу, и т.п.



Достоинства AJAX

- Возможность создания удобного Web-интерфейса
- Активное взаимодействие с пользователем
- Частичная перезагрузка страницы, вместо полной
- Удобство использования

При использовании AJAX нет необходимости обновлять каждый раз всю страницу, так как обновляется только ее конкретная часть. Это намного удобнее, так как не приходится долго ждать, и экономичнее, так как не все обладают безлимитным интернетом.

Правда в этом случае, разработчику необходимо следить, чтобы пользователь был в курсе того, что происходит на странице. Это можно реализовать с использованием индикаторов загрузки, текстовых сообщений о том, что идёт обмен данными с сервером.

Необходимо также понимать, что не все браузеры поддерживают AJAX (старые версии браузеров и текстовые браузеры). Плюс Javascript может быть отключен пользователем. Поэтому, не следует злоупотреблять использованием технологии и прибегать к альтернативным методам представления информации на Web-сайте.

AJAX - это интеграции технологий

AJAX-приложение состоит как минимум из двух частей.

Первая выполняется в браузере и написана, как правило, на JavaScript.

Вторая - находится на сервере и написана, например, на Ruby, Java или PHP.

Между этими двумя частями происходит обмен данными через XMLHttpRequest.

HTML, CSS для подачи и стилизации информации.

DOM-модель, операции над которой производятся javascript на стороне клиента, чтобы обеспечить динамическое отображение и взаимодействие с информацией.

XMLHttpRequest для асинхронного обмена данными с веб-сервером.

JSON часто используется для обмена данными, однако любой формат подойдет, включая форматированный HTML, текст, XML.

Асинхронная и синхронная модель в AJAX

В синхронной модели браузер отправляет запрос на сервер и висит, ждет, пока тот совершит всю необходимую работу. Сервер выполняет запросы к базе данных, заворачивает ответ в необходимый формат и выводи его. Браузер. получив ответ, вызывает функцию показа.

Все процессы выполняются последовательно, один за другим.

В асинхронной модели запрос отсылается, и можно заняться чем-то другим. Когда запрос выполнился – запускается заранее подготовленная программистом функция показа сообщения сервера.

Обмен данными

Для того, чтобы осуществлять обмен данными, на странице должен быть создан объект XMLHttpRequest, который является своеобразным посредником между Браузером пользователя и сервером. С помощью XMLHttpRequest можно отправить запрос на сервер, а также получить ответ в виде различного рода данных.

```
// 1. Создаём новый объект XMLHttpRequest
var xhr = new XMLHttpRequest();
// 2. Конфигурируем его: GET-запрос на URL 'phones.html'
xhr.open('GET', 'phones.html', false);
// 3. Отсылаем запрос
xhr.send();
// 4. Если код ответа сервера не 200, то это ошибка
if (xhr.status != 200) {
  // обработать ошибку
  alert( xhr.status + ': ' + xhr.statusText ); // пример вывода: 404: Not Found
} else {
  // вывести результат
  alert( xhr.responseText ); // responseText -- текст ответа.
}
```

jQuery — быстрый и надежный способ программирования на JavaScript и Ajax. Набор функций Ajax, доступных через jQuery, значительно упрощает Ajax-разработку, позволяя писать меньше кода и применять дополнительные методы и обработчики событий, охватывающие большинство ситуаций.

С jQuery требуется минимальное количество кода даже при разработке сложных функций, так что процесс идет намного быстрее. Благодаря этим инструментам Ajax становится более практичным способом разработки Web-сайта или приложения в сжатые сроки.

jQuery - это, по существу, библиотеки JavaScript, которая упрощает разработку JavaScript. Она минимизирует код, который нужно написать, благодаря множеству встроенных возможностей, обычно требующих создания специальных функций или объектов.

jQuery.ajax

```
$.ajax({
  url: 'ajax/test.html',
  success: function(data){
    alert( "Прибыли данные: " + data );
  },
  error: function () {
          alert('Error');
      }
});
```

async - асинхронность запроса, по умолчанию true
cache - вкл/выкл кэширование данных браузером, по умолчанию true
contentType - по умолчанию "application/x-www-form-urlencoded"
data - передаваемые данные строка иль объект
dataFilter - фильтр для входных данных
dataType - тип данных возвращаемых в callback функцию (xml, html, script, json, text, _default)
global - тригер - отвечает за использование глобальных AJAX Event'ов, по умолчанию true
ifModified - тригер - проверяет были ли изменения в ответе сервера, дабы не слать еще запрос, по умолчанию false
jsonp - переустановить имя callback функции для работы с JSONP (по умолчанию генерируется на лету)
processData - по умолчанию отправляемые данный заворачиваются в объект, и отправляются как "application/x-www-form-urlencoded", если надо иначе - отключаем
scriptCharset - кодировочка - актуально для JSONP и подгрузки JavaScript'ов
timeout - время таймаут в миллисекундах
type - GET либо POST
url - url запрашиваемой страницы

Локальные AJAX Event'ы:

Обработчики событий, которые происходят в определенные моменты выполнения каждого ajax-запроса.

ajaxStart — Данный метод вызывается в случае когда побежал AJAX запрос, и при этом других запросов нету

beforeSend — Срабатывает до отправки запроса, позволяет редактировать XMLHttpRequest. Локальное событие

ajaxSend — Срабатывает до отправки запроса, аналогично beforeSend

success — Срабатывает по возвращению ответа, когда нет ошибок ни сервера, ни вернувшихся данных. Локальное событие

ajaxSuccess — Срабатывает по возвращению ответа, аналогично success

error — Срабатывает в случае ошибки. Локальное событие

ajaxError — Срабатывает в случае ошибки

complete — Срабатывает по завершению текущего AJAX запроса (с ошибкои или без — срабатывает всегда).

ajaxStop — Данный метод вызывается в случае когда больше нету активных запросов