Введение в JavaScript

Лекция 16

УНЦ «Инфоком»
Самара
2013

NetCracker®

План занятия

- Общие сведения о JavaScript
- Основы синтаксиса
- Основы работы с объектами
- Основы работы с документом и браузером

JavaScript

- JavaScript – это язык программирования
  - Интерпретируемый
  - Объектно-ориентированный
  - Со слабой типизацией
  - С динамической типизацией
  - С автоматическим управлением памятью
  - Вместо наследования классов используются прототипы объектов
  - Функции являются объектами

Области применения

- Скриптовые фрагменты серверных приложений
- Виджеты
- Прикладное программное обеспечение
- Букмарклеты (небольшие приложения, размещаемые в закладках браузера)
- Пользовательские скрипты в браузерах

Области применения

- Программы на стороне клиента в web-приложениях
  - Код встраивается в HTML-страницу
  - Код выполняется браузером
  - Возможно изменение структуры страницы, её элементов и их параметров
  - Изменения могут происходить без перезагрузки страницы
  - Могут выполняться дополнительные действия
    - Взаимодействие с браузером
    - Взаимодействие с сервером
    - Бизнес-логика в целом

Здравствуй, мир!!!

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Проверка JS</title>
  </head>
  <body>
    <h1>
      <script>
        document.write("Hello, world!!!");
      </script>
    </h1>
  </body>
</html>
```

Размещение в HTML-документе

- Тег <script>

```
<html>
  <body>
    <p id="demo">This is a paragraph.</p>
    <script type="text/javascript">
      document.getElementById("demo").innerHTML=Date();
    </script>
  </body>
</html>
```

- Атрибуты тегов (обработка событий)

```
<a href="delete.jsp" onclick="return confirm('Вы уверены?');">Удалить</a>
```

- Гипертекстовые ссылки

```
<a href="javascript:document.mainform.submit();">Отправить</a>
```

Тег <script>

- Указание языка
  - По умолчанию – именно JavaScript
  - В явном виде – с помощью атрибута
    - Устаревший вариант
      `language="JavaScript"`
    - Современный вариант
      `type="text/javascript"`
- Две формы кода
  - Код размещается внутри тега
  - Код размещается в отдельном файле (обычно с расширением `js`), файл указывается с помощью атрибута `src`
  - Второй способ «перекрывает» первый
- Типичное использование
  - В заголовке документа – для описания функций и предварительных действий
  - В теле документа – для условной генерации документа

Обработка браузером

- Обычно код интерпретируется
  - Возможны Just-In-Time решения
- Код выполняется по мере обнаружения
  - Обращаться к элементам документа, описанным в документе позднее – не лучшая мысль
  - Однако обращаться к позднее описанным сущностям из функций можно, если сущности уже будут существовать на момент выполнения функции
- Возможно возникновение ошибок в ходе выполнения
  - Блок скрипта не завершит работу
  - В консоли браузера появится сообщение об ошибке

Собственно, код

- JavaScript чувствителен к регистру!
  - HTML – нечувствителен к регистру (хотя есть рекомендации)
  - Соседство атрибутов HTML и событий/методов JavaScript может фрустрировать (например, `onClick` и `onclick`)
- Выполнение кода
  - Функции выполняются при явном вызове
  - Просто код в теге `<script>` выполняется по мере обнаружения
  - Код, указанный в атрибутах тегов как обработка событий, выполняется при возникновении событий
  - Код, указанный в ссылках, выполняется при срабатывании ссылки

Комментарии

```
<html>
  <body>
    <script>
      document.write("Это будет выведено.");
      // document.write("А это не будет выведено...");
      /*
      document.write("И это не будет выведено!");
      document.write("И это тоже не будет, надо же...");
      */
      <!-- document.write("И это не будет?!!");
    </script>
  </body>
</html>
```

Комментарии

- Однострочные
  - От символов `//` до конца строки
  - От символов `<!--` до конца строки

```
<script type="text/javascript">
<!--
document.write("Вы не увидите никаких следов кода,");
document.write("если ваш браузер не поддерживает JavaScript!!!");
//-->
</script>
```

- Многострочные
  - Начало `/*`
  - Конец `*/`
  - Не могут быть вложенными

Литералы и переменные

- Литералы
  - Числовые
    `10  010 0x10 1.1 .1 1. 1e1`
  - Строковые
    `"Строка" 'Строка' "I'm a string" '\'' "\""`
- Переменные
  - Могут объявляться с помощью ключевого слова `var`
    - `var a = 5, b = " лет";`
    - Если объявление вне функции – переменная глобальная, иначе – локальная
  - Могут объявляться и без ключевого слова `var`
    - `c = "И так тоже можно"`
    - Переменная всегда глобальная…
  - Объявление переменных «на лету» + их глобальность + чувствительность к регистру = отличный способ выстрелить себе в ногу
  - Тип переменной определяется её содержимым
  - Приведение типов автоматическое

Операторы

Приоритет	Оператор	Запись
1	Доступ к элементу	`. []`
	Создание объекта	`new`
2	Вызов функции	`()`
3	Инкремент (две формы)	`++`
	Декремент (две формы)	`--`
4	Логическое отрицание	`!`
	Побитовое отрицание	`~`
	Унарный плюс	`+`
	Унарный минус	`-`
	Тип объекта	`typeof`
	Вычисление выражения без возврата значения	`void`
	Удаление свойства объекта	`delete`

Операторы

Приоритет	Оператор	Запись
5	Умножение	`*`
	Деление	`/`
	Остаток от деления	`%`
6	Сложение, конкатенация	`+`
	Вычитание	`-`
7	Побитовые сдвиги	`<< >> >>>`
8	Сравнение	`< <= > >=`
	Проверка наличия свойства в объекте	`in`
	Проверка «типа» объекта	`instanceof`
9	Равенство и строгое (с типом) равенство	`== != === !===`

Операторы

Приоритет	Оператор	Запись
10	Побитовое И	`&`
11	Побитовое исключающее ИЛИ	`^`
12	Побитовое ИЛИ	`|`
13	Логическое И	`&&`
14	Логическое ИЛИ	`||`
15	Тернарный условный оператор	`?:`
16	Присваивания	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
17	Разделитель последовательности выражений	`,`

Управление порядком выполнения

- Операторные скобки
  - `{...}`
- Ветвление
  - `if (условие) инструкция;`
  - `if (условие) инструкция; else инструкция;`
  - Нелогические значения `0`, `""`, `null`, `undefined` и `NaN` считаются ложью, всё остальное – истиной

Управление порядком выполнения

- Цикл с предусловием
  - `while(условие) инструкция;`
- Цикл с постусловием
  - `do { ... } while(условие);`
- Итерационный цикл
  - `for(инициализация; условие; действие) инструкция;`
- Цикл по элементам
  - `for(переменная in объект) инструкция;`

Управление порядком выполнения

- Прерывание блока
  - `break;`
  - `break имяМетки;`
- Переход на следующую итерацию
  - `continue;`
  - `continue имяМетки;`
- Возврат из метода или обработчика
  - `return;`
  - `return значение;`

Управление порядком выполнения

- Блок переключателей
  - `switch (выражение) {`
    `case значение: инструкции;`
    `...`
    `default: инструкции}`
  - Для сравнения используется строгое равенство
  - Можно работать со строками
  - Предложений `case` может быть много
  - Для прерывания используется `break`
  - Есть предложение `default`

Функции

- Объявление функции (function declaration)

```
function имя(парамр1, ... параметрN) {
  тело функции
}
```

- Формальные и фактические параметры различаются
- Внутри функции объявление переменных с `var` даёт локальные переменные
- Функции создаются предварительно, до выполнения кода

```
writeLine("Итого:");
writeLine(745);
function writeLine(line) {
  document.write(line);
  document.write("<br/>");
}
```

Функции – это объекты

- Функции – это объекты, у которых есть имя и значение, и которые можно передавать в функции

```
writeLine(writeLine);
/* Будет выведено следующее:
function writeLine(line) { document.write(line); document.write("<br/>"); }
*/
```

- Функция-выражение (function expression)

```
function(параметр1, ..., параметрN) {
  тело функции
}
```

  - Полученную функцию можно присвоить в переменную
  - Функции создаются не заранее, а когда до них доходит выполнение
  - Условное объявление функций (переменная объявляется заранее, реализация – потом)
  - Лучше такой формой не злоупотреблять

Функции – это объекты

- Можно вызывать функции, полученные в качестве объектов, и передавать им параметры

```
function showResult(a, b, calculator) {
  document.write("Операнд 1: " + a + "<br/>");
  document.write("Операнд 2: " + b + "<br/>");
  document.write("Результат: " + calculator(a, b) + "<br/>");
}
function sum(a, b) {
  return a + b;
}
showResult(5, 7, sum); //
showResult(8, 3, function (a, b) { return a - b; });
```

Объекты

- Являются по сути ассоциированными массивами (картами, map)
  - ключи – имена свойств (только строки)
  - значения – значения свойств (любые типы, **включая функции**)
- Доступ к свойствам
  - `ссылка.имяСвойства`
  - `ссылка["имяСвойства"]`
- Свойства добавляются и исключаются динамически
- Переменные хранят ссылки на объекты
- Свойства объектов являются переменными

Операции со свойствами

```
var employee = {}; // Создание пустого объекта

employee.name = "King"; // Добавление свойств
employee.salary = 5000;

document.write(employee.name + ": " + employee.salary); // Чтение свойств

delete employee.salary // Удаление свойства

for (p in employee) { // Ещё одно чтение свойств
  document.write(typeof(p) +  " " + p + " " + employee[p]);
}
/* Будет выведено
King: 5000
string name King
/*
```

Операции со свойствами

```
// Литеральное создание объекта
var rectangle = {
  width: 200,
  height: 300,
  getArea: function() { return this.width * this.height; }
}; // this в методах при обращении к свойствам обязателен!!!

// Изменение свойств
rectangle.width = rectangle.height = 5;
rectangle.getPerimeter = function() {
  return 2 * (this.width + this.height);
};

// Проверка работы
document.write(rectangle.getArea() + "<br/>");
document.write(rectangle.getPerimeter() + "<br/>");
```

Функции-конструкторы

- Конструкторами объектов являются функции
  - Их принято называть с большой буквы
  - Вызываются с помощью **new**
  - Могут иметь параметры
  - Формируют объект, используя слово **this**
  - Возвращают ссылку на созданный объект

```
function Employee(name, salary) {
  this.name = name;
  this.salary = salary;
  this.fired = false;
}
var king = new Employee("King", "5000");
```

Встроенные объекты

- Конструкторы
  - Math
  - Date
  - RegExp
  - Function
  - Array
  - ...
- «Обёртки»
  - String
  - Number
  - Boolean

Наследование

- Базовым является не наследование классов, а наследование объектов
- Родительский объект называется прототипом
- Механизмы наследования заметно отличаются от классического ООП с классами
- Можно эмулировать почти что классические классы со всеми их возможностями и проблемами

Обработка исключений

- Обработка исключений

```
try {
  // код, потенциально выбрасывающий исключения
}
catch (e) {
  // код обработки ошибки
}
finally {
  // код, всегда выполняющийся в конце
}
```

- Блока `catch` или `finally` может не быть
- Для непользовательских методов ошибка обычно «имеет тип» `Error`, свойства которого могут отличаться в зависимости от браузера

Выбрасывание исключений

- Метод вправе выбросить своё исключение

```
throw ссылкаНаОбъектИсключения;
```

- Можно выбросить вообще любой объект

```
throw 12345;
```

- Но лучше выбрасывать что-нибудь вразумительное

```
throw {name: "OhShit!", message: "Ohhhh"}
throw new Error("Oooops!");
```

Смена контекста

- Смена текущего объекта

```
var employee = {name: "King", salary: 5000};
with (employee) {
  document.write(name + ": " + salary);
}
```

- Смена контекста this у функции/метода

```
function raiseSalary(addition, factor) {
  this.salary = addition + this.salary * factor;
}
var petrov = {position: "Salesman", salary: 1000};
var president = {name: "King", salary: 5000};
raiseSalary.call(petrov, 100, 1.1);
raiseSalary.apply(president, [0, 1.2]);
```

Пользовательские массивы

- Одномерные
  - Но можно создать массив массивов, в т.ч. непрямоугольный
  - Обращение по индексу с помощью оператора `[]`
  - Нумерация элементов с 0
  - Есть поле `length`, хранящее количество элементов
- Способы создания
  - Пустой массив
    ```
    var a1 = new Array();
    ```
  - Массив с заданным количеством элементов
    ```
    var a2 = new Array(10);
    ```
  - Массив с заданными элементами
    ```
    var a3 = new Array(10, "и это не длина", 5.5, '!');
    ```
  - Литеральная форма с заданными элементами
    ```
    var a4 = [10, "и это не длина", 5.5, '!'];
    ```

Пользовательские массивы

- Динамические
  - Изменение значения `length`
  - Добавление новых элементов
  - Явно не указанные элементы получают значение `undefined`

```
<script>
  var a = [1, 2];
  a[5] = 5;
  document.write(a[4] + "<br/>");
  document.write(a[5] + "<br/>");
  a.length = 2;
  a.length = 5;
  document.write(a[4] + "<br/>");
  document.write(a[5] + "<br/>");
</script>
```

Сортировка пользовательских массивов

- Выполняется с помощью метода `sort()` массива
- По умолчанию – это сортировка в лексикографическом порядке
- Если требуется другой порядок, то следует задать свой критерий сравнения в виде функции

```
<script>
  var a = [1, 2, 15, 23];
  a.sort();
  document.write(a + "<br/>"); // 1,15,2,23
  function compareNumeric(a, b) {
    return a - b;
  }
  a.sort(compareNumeric);
  document.write(a + "<br/>"); // 1,2,15,23
</script>
```

Дополнительные методы пользовательских массивов

Инвертирование порядка элементов	`reverse()`
Добавление элемента в конец	`push(element)`
Удаление элемента из конца	`pop()`
Добавление элемента в начало	`unshift(element)`
Удаление элемента из начала	`shift()`
Объединение в строку с указанием разделителя	`join(separator)`
Удаление и вставка элементов	`splice(start, [deleteCount, add1, ..., addN])`
Копирование части массива	`slice(begin, end)`

Дополнительные методы пользовательских массивов

```
<html>
  <body>
    <script>
      var a = [1, 2, 15, 23];
      a.reverse();
      document.write(a + "<br/>"); // 23,15,2,1
      a.push("добавка 1");
      document.write(a + "<br/>"); // 23,15,2,1,добавка 1
      a.pop();
      document.write(a + "<br/>"); // 23,15,2,1
      a.unshift("добавка 2");
      document.write(a + "<br/>"); // добавка 2,23,15,2,1
      a.shift();
      document.write(a + "<br/>"); // 23,15,2,1
      //...
```

Дополнительные методы пользовательских массивов

```
        // ...
        var str = a.join("; ");
        document.write(str + "<br/>"); // 23; 15; 2; 1
        var b = str.split("; ");
        document.write(b + "<br/>"); // 23,15,2,1
        a.splice(2, 1, "Замена");
        document.write(a + "<br/>"); // 23,15,Замена,1
        var c = a.slice(1, a.length);
        c[0] = "И что будет?";
        document.write(a + "<br/>"); // 23,15,Замена,1
        document.write(c + "<br/>"); // И что будет?,Замена,1
    </script>
  </body>
</html>
```

Виды объектов

- Встроенные
  - По сути – библиотеки и базовые объекты
- Пользовательские
  - Всё, что создаёт пользователь-программист
- Серверные
  - Определяют и предоставляют взаимодействие с сервером
- Клиентские
  - Определяют и предоставляют взаимодействие с браузером и документом

Клоуны BOM и DOM

- Browser Object Model
  - Объектная модель для взаимодействия с браузером
  - В настоящий момент не стандартизована
  - Базовый объект – `window`
- Document Object Model
  - Объектная модель для взаимодействия с документом
  - В целом стандартизирована
  - Базовый объект – `document`

Объект window

- Глобальный объект
- Все объявляемые переменные и объекты становятся его свойствами
- Содержит либо напрямую информацию о документе, либо ассоциированный массив (карт, map) фреймов (frames)
- Имеет свои свойства, методы и события

Взаимодействие с пользователем

```
// Окно с сообщением
alert("Случилось страшное!!!");

// Окно с запросом на подтверждение
// Возвращает true или false
confirm("Хотите поговорить об этом?..");

// Окно со вводом строки
// Возвращает введённую строку
prompt("Вам слово!", "говорите сюда");
```

Создание и закрытие новых окон

- Создание
  - Параметры задаются строкой
  - Поведение зависит от браузера
  - Возвращается ссылка на объект окна

```
var ncwin = window.open('http://www.netcracker.com',
 'Сайт компании', 'directories=no,height=300,location=no,' +
 'menubar=no,resizable=yes,scrollbars=yes,' +
 'status=no,toolbar=no,width=300');
```

- Закрытие

```
ncwin.close();
```

Запуск новых потоков

- Запуск нового потока
  - `setTimeout("код", времяВМиллисекундах)`
  - Возвращает ссылку на поток
- Остановка запущенного разового потока
  - `clearTimeOut(ссылкаНаПоток)`
- Периодический запуск нового потока
  - `setInterval("код", времяВМиллисекундах)`
  - Возвращает ссылку на поток
- Остановка запущенного периодического потока
  - `clearInterval(ссылкаНаПоток)`

Свойства window

- `location` – объект «типа» URL, текущий адрес
  - `href` – собственно URL
  - Набор свойств, соответствующих фрагментам URL
  - Методы
  - `replace()`
  - `reload()`
- `history` – объект, хранящий историю переходов по страницам
  - `forward()`
  - `back()`
  - `go(количествоСтраниц)`
- `navigator` – объект, хранящий данные о браузере
- `opener` – ссылка на родительское окно

Объект document

- В нём и живёт DOM
- Это тоже ассоциированный массив специфического вида
- Все теги получают в соответствие объект
  - Имя объекта определяется значением атрибута `name` тега
  - Атрибуты тегов становятся свойствами соответствующих объектов
  - Не все свойства объектов соответствуют атрибутам тегов
- Объекты формируются по ходу чтения документа браузером
- Некоторые виды объектов объединяются в дополнительные массивы (формы, ссылки и т.д.)
- У объектов могут быть события
- У объектов могут быть дополнительные методы

Прямая запись в документ

- Методы
  - `document.write(значение)`
  - `document.writeln(значение)`
- Используются для генерирования (в т.ч. условного) содержимого документа
- Можно использовать только если документ ещё «открыт»
  - Для загружаемых документов – документ ещё не до конца прочитан браузером
  - Для документов в динамически открытых окнах – после вызова `document.open()` и до вызова `document.close()`
- Если документ уже закрыт, то его можно изменять только через объекты и управление ими

События объектов

- Обычно именуются `onчто-нибудь`
- Обработчик можно указать прямо в HTML-коде
- Обработчик можно задать программно
- Возврат из некоторых обработчиков null означает прекращение обработки

```
<html>
 <body>
  <p
onMouseOver="document.getElementById('addition').innerHTML = 'А он есть...'"
onMouseOut="document.getElementById('addition').innerHTML = ''">
    Видишь суслика?..</p>
  <p id="addition"></p>
  <img id="picture" src="http://turbo-suslik.ru/images/suslik.jpg"
       width="150px" height="150px"/>
  <script>
    document.getElementById("picture").onclick = function() {
      alert("Вот же он, суслик!")};
  </script>
 </body>
</html>
```

Динамическое изменение DOM

```
<html>
  <body>
    <table border="1px" id="table">
      <head id="head"> <td>Фамилия</td> <td>Имя</td> </head>
      <tr> <td>Иванов</td> <td>Иван</td> </tr>
    </table>
    <form name="mainform">
      Фамилия:
      <input type="text" name="sname"/><br/>
      Имя:
      <input type="text" name="fname"/><br/>
      <input type="submit" name="addition" value="Добавить"
         onClick="appendToTable(document.mainform.fname.value,
document.mainform.sname.value); return false;"/>
    </form>
```

Динамическое изменение DOM

```
    <script>
      function appendToTable(fname, sname) {
        var table = document.getElementById("table");
        var row = document.createElement("tr");
        var surname = document.createElement("td");
        var firstname = document.createElement("td");
        surname.innerHTML = sname;
        firstname.innerHTML = fname;
        row.appendChild(surname);
        row.appendChild(firstname);
        table.appendChild(row);
      }
    </script>
  </body>
</html>
```

Работа с формами

- Доступ к форме
  - По имени как к свойству документа
  - В массиве `document.forms`
- Доступ к элементам
  - По имени как к свойству формы
  - В массиве `elements` формы
- Атрибуты тегов элементов формы и самой формы – свойства соответствующих объектов
- События формы
  - `onsubmit`
  - `onreset`
- Методы формы
  - `submit()`
  - `reset()`

Программирование гиперссылок

- Доступ к гиперссылкам
  - По имени/идентификатору
  - В массиве `document.links[]`
  - Объект имеет «тип» URL
- Объект ссылки имеет свойства и события
- Ссылка может иметь программный вид
  - `javascript:код`
  - результат работы кода может быть показан в браузере
  - Результатом считается результат последнего выражения
  - Приём `void(0);`
  - Так можно не только в ссылке, но и в `action` у форм

Использование ссылок

```
<html>
  <body>
    <table border="1px" id="table">
      <head id="head"> <td>Фамилия</td> <td>Имя</td> </head>
      <tr> <td>Иванов</td> <td>Иван</td> </tr>
    </table>
    <form name="mainform">
      Фамилия: <input type="text" name="sname"/><br/>
      Имя: <input type="text" name="fname"/><br/>
    </form>
    <a href="javascript:appendToTable(document.mainform.fname.value,
                                      document.mainform.sname.value);
            void(0);">Добавить</a>
    <a href="javascript:document.mainform.submit();">
      Отправить данные</a>
```

Некоторые замечания

- Часто JavaScript воспринимается как набор приёмов по программированию клиентской части
- Это не так
- JavaScript – интересный и богатый язык с широкими возможностями
- Он просто недооценён сообществом
- И детальное его изучение требует значительного времени…

Что осталось за бортом

- Объектная модель, наследование, паттерны и т.д.
- Детали работы с DOM и BOM
- Свойства, методы и события объектов, соответствующих HTML-тегам
- Встроенные объекты и их возможности
- Регулярные выражения
- Работа с графикой
- Работа с анимацией
- Возможности в сочетании с CSS
- Работа с Cookie
- …

Где почитать

- http://learn.javascript.ru/
- http://citforum.ru/internet/javascript/
- https://developer.mozilla.org/en/JavaScript/Reference/
- http://www.w3schools.com/js/default.asp
- …
- Несть им числа!

Спасибо за внимание!

NetCracker®
NetCracker®