

JavaScript

ОСНОВЫ

Что нужно знать?

- Широко используемая технология и число прецедентов её применения растёт
 - Web-page, AJAX, Web 2.0
 - Увеличение числа приложения имеющих Web-интерфейс
- Интересные и необычные особенности
 - Каждая функция это объект
 - У объектов нет классов
 - Широкие возможности изменения поведения объектов

Много проблем с безопасностью

- Нет статических типов – типы и переменные могут меняться во время работы программы
- Сложно предсказать поведение программы заранее

Возможности изменения поведения объектов

- Всегда можно добавить новое поле объекту

Возможности изменения поведения объектов

- Всегда можно добавить новый метод объекту

Возможности изменения поведения объектов

- Вызвать функцию от имени объекта

История JavaScript

- Разработан Брендан Айк (Brendan Eich) в компании Netscape
 - Скриптовый язык для Navigator 2
- Позже стандартизирован для браузерной совместимости
 - ECMAScript Editor 3 (JavaScript 1.5)
 - ES 5 – актуальная версия
- Связан с Java только по названию
 - Название было маркетинговым ходом
- Доступные версии JavaScript
 - Spidermonkey
 - Rhino
 - V8
 - ...



Мотивации к созданию JavaScript

- Netscape 1995
 - Занимает 90% рынка браузеров
 - Возможность сделать скрипты для HTML
- Потребности к использованию JavaScript
 - Проверка форм
 - Украшение и специальные эффекты на страницах
 - Динамическая манипуляция с контентом
 - Исполнение некоторых действий на машинах клиентов

Пример (вычисления)

```
<html>
...
<p>
...
</p>
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>
...
</html>
```

Пример (события)

```
<script type="text/JavaScript">
  function whichButton(event) {
    if (event.button==1) {
      alert("You clicked the left mouse button!")
    } else {
      alert("You clicked the right mouse button!")
    }
  }
</script>
...
<body onmousedown="whichButton(event)">
  ...
</body>
```

Другие возможные события: onLoad, onMouseMove, onKeyPress, onUnload,
...

Пример работы со страницей

- Возможности
 - `createElement(elementName)`
 - `createTextNode(text)`
 - `appendChild(newChild)`
 - `removeChild(node)`
- Пример добавления нового маркированного списка:

```
var list = document.getElementById('list1');  
var newItem = document.createElement('li');  
var newText = document.createTextNode(text);  
list.appendChild(newItem);  
newItem.appendChild(newText);
```

Этот скрипт
изменяет DOM

Цели

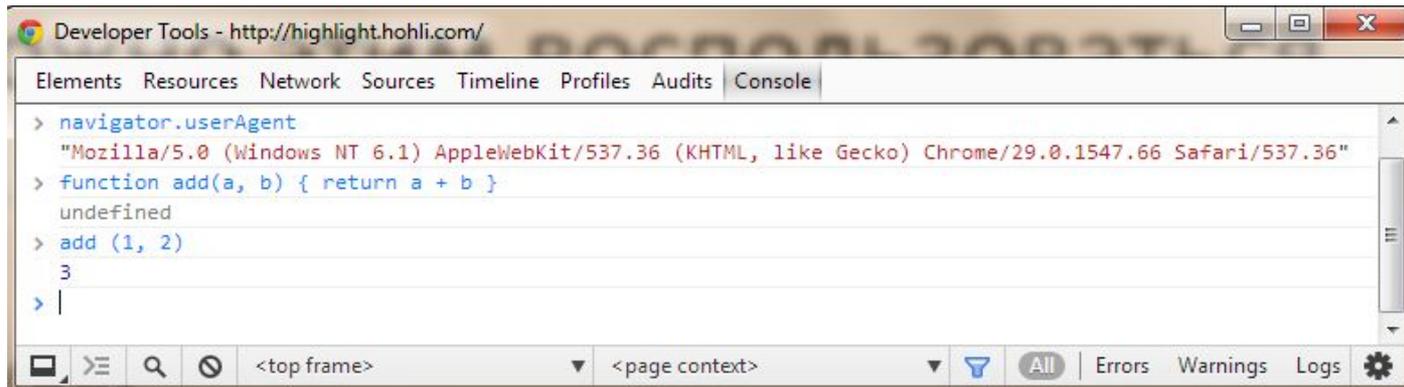
- Сделать простой процесс copy/paste полезных участков кода
- Терпеть незначительные ошибки (нет точки с запятой)
- Простой процесс обработки событий (onclick, onmousedown, ...)
- Возможность выбора парадигмы:
 - Процедурная абстракция
 - ООП через прототипы

База языка

- Чувствительность к регистру
 - А и а разные идентификаторы
- Выражения (statements) оканчиваются или возвращением значения или (;)
 - $x=x+1$; тоже что и $x=x+1$
 - Точку с запятой лучше ставить для уменьшения числа ошибок
- Блок
 - Группа выражений – {...}
 - Не выделяет отдельное пространство имен (scope)
- Переменные
 - Определение переменных с использованием var
 - Определение без слова var обязывает установить значение при первом использовании
 - При таком определении переменная будет иметь глобальную область видимости

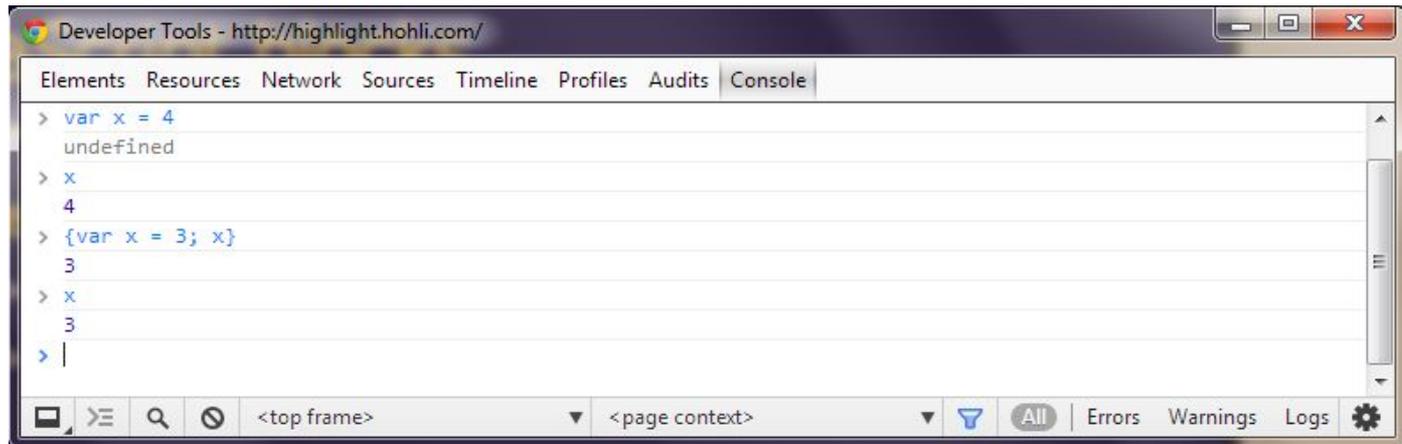
Как можно ЭТИМ ВОСПОЛЬЗОВАТЬСЯ

- V8
- Event loop
 - Вводим выражение
 - Прерывается выполнение event-loop
 - Отображается значение
 - Продолжается выполнение event-loop
- Пример



JavaScript блок

- Для группировки, но не отделяет пространство имён



```
Developer Tools - http://highlight.hohli.com/
Elements Resources Network Sources Timeline Profiles Audits Console
> var x = 4
undefined
> x
4
> {var x = 3; x}
3
> x
3
> |
```

- Не блок в смысле других языков программирования
 - Только вызов функции и выражение with отделяет область видимости

JavaScript ТИПЫ

- Boolean
 - Два значения true и false
- Number
 - 64-битное число с плавающей точкой, похоже на java double и Double
 - Нет целочисленного типа
 - Специальные значения NaN (Not a Number), Infinity
- String
 - Последовательность 0 или больше символов Unicode
 - Нет типа символ, только строки длиной 1
 - Литерал строки отделяется либо (') либо (") кавычкой
- Специальные значения
 - null и undefined
 - typeof(null) = object
 - typeof(undefined) = undefined

Объекты

- Объекты это именованные коллекции
 - Проще всего представить в виде ассоциированного массива
 - Можно определить множество пар имя-значение
 - `objBob = {name: "Bob", grade: 'A', level: 3};`
 - Новое свойство может быть добавлено в любое время
 - `objBob.fullname = 'Robert';`
 - Методы могут ссылаться на `this`
- Массивы и функции тоже объекты
 - Свойствами объекта могут быть функции
 - Функции это те же объекты с методом `()`
 - `function max(x,y) { if (x>y) return x; else return y;};`
 - `max.description = "return the maximum of two arguments";`

ФУНКЦИИ

- В теле функции могут быть
 - Локальные переменные
 - Вложенные (inner) функции
- Передача параметров
 - Базовые типы передаются по значению, объекты по ссылке
- Вызов функции можно сделать с любым числом аргументов
 - `functionname.length` – число определённых аргументов
 - `functionname.arguments.length` – число аргументов с которыми была вызвана функция

ФУНКЦИИ

- Анонимные функции (выражение из функции)
 - `(function (x,y) {return x+y}) (2,3);`
- Замыкания и функции возвращающие функции
 - `function CurAdd(x){ return function(y){return x+y} };`
- Анонимные функции могут быть функциями обратного вызова (callback)
 - `setTimeout(function() { alert("done"); }, 10000)`

ФУНКЦИИ

- **Функции возвращающие функции**

```
function CurriedAdd(x){ return function(y){ return x+y} };
```

```
g = CurriedAdd(2);
```

```
g(3)
```

- **Переменное число аргументов**

```
function sumAll() {  
    var total=0;  
    for (var i=0; i< sumAll.arguments.length; i++)  
        total+=sumAll.arguments[i]; return total);  
}  
sumAll(3,5,3,5,3,2,6)
```

Использование анонимных функций

- Анонимные функции широко применяются как функции обратного вызова

```
setTimeout(function() { alert("done"); }, 10000)
```

// putting alert("done") in function delays evaluation until call

- Можно сделать блок с отделённым

```
□ var u = { a:1, b:2 }  
  var v = { a:3, b:4 }  
  (function (x,y) {  
    var tempA = x.a;  
    var tempB =x.b; //local variables  
    x.a=y.a;  
    x.b=y.b;  
    y.a=tempA;  
    y.b=tempB  
  }) (u,v) // Объекты здесь передаются по ссылке
```

Лямбда выражения

- Выражение

$$- x + y \qquad x + 2y + z$$

- Функции

$$- \lambda x.(x + y) \qquad \lambda z.(x + 2y + z)$$

- Приложение

$$- \lambda x.(x + y)(3) \qquad = 3 + y$$

$$- \lambda z.(x + 2y + z)(5) \qquad = x + 2y + 5$$

Порядок выполнения

- Отдаём функцию f , получаем суперпозицию $f \circ f$
– $\lambda f. \lambda x. f(f x)$
- Как это должно работать

$$\begin{aligned} & (\lambda f. \lambda x. f(f x)) (\lambda y. y+1) \\ &= \lambda x. (\lambda y. y+1) ((\lambda y. y+1) x) \\ &= \lambda x. (\lambda y. y+1) (x+1) \\ &= \lambda x. (x+1)+1 \end{aligned}$$

Те же процедуры в синтаксисе Lisp

- Отдаём функцию f , получаем суперпозицию $f \circ f$

```
(lambda (f) (lambda (x) (f (f x))))
```

- Как это работает

```
((lambda (f) (lambda (x) (f (f x)))) (lambda (y) (+ y 1)))
```

```
= (lambda (x) ((lambda (y) (+ y 1)) ((lambda (y) (+ y 1)) x))))
```

```
= (lambda (x) ((lambda (y) (+ y 1)) (+ x 1)))
```

```
= (lambda (x) (+ (+ x 1) 1))
```

Тоже в JavaScript

- Отдаём функцию f , получаем суперпозицию $f \circ f$

```
function (f) {  
  return function (x) {  
    return f(f(x));  
  };  
}
```

- Как это должно работать

```
(function (f) {  
  return function (x)  
{  
    return f(f(x));  
};})(function (y) {  
  return y + 1;  
})
```



```
function (x) {  
  return ((x + 1) + 1);  
}
```

Особенности использования объектов

- Использование функции – конструктора

```
function car(make, model, year) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
}
```

- Объект имеет прототип, который можно ИЗМЕНИТЬ

```
var c = new car("Ford", "Taurus", 1988);  
car.prototype.print = function () {  
    return this.year + " " + this.make + " " + this.model;  
}  
c.print();
```

Особенности this в JavaScript

```
var x = 5;
var y = 5;
function f() {
    return this.x + y;
}
var o1 = {x : 10}
var o2 = {x : 20}
o1.g = f;
o2.g = f;
o1.g()
o2.g()
```

Оба свойства o1.g и o2.g ссылаются на одну и ту же функцию, но результаты её выполнения разные

□ ?

□ ?

Ещё о this

- В большинстве случаев this указывает на объект который содержит функцию как метод

- Пример

```
var o = {  
  x : 10,  
  f : function () {  
    return this.x  
  }  
}  
o.f();
```

- this разрешится динамически во время исполнения метода

Особенности для вложенных методов

```
var o = {  
  x: 10,  
  f : function () {  
    function g () {  
      return this.x  
    };  
    return g ();  
  }  
};  
o.f ()
```

Функция g взмёт глобальный объект как this!
(Обычно это window)

Особенности закреплённые в стандарте

- Управляемый стек памяти для функций
 - Параметры функций и локальные переменные
- Сборщик мусора (Garbage collector)
 - Автоматическое освобождение памяти
- Замыкания
- Исключения
- Объектная модель
 - Динамическое расширение, инкапсуляция и наследование через прототипы
- Многопоточность
 - Можно делать одновременно несколько задач (JavaScript – всегда однопоточный)

Управляемый стек памяти

- Локальные переменные существуют только внутри функции

```
function f(x) {  
    var y = 3;  
    function g(z) {  
        return y+z;  
    };  
    return g(x);  
}  
var x = 1;  
var y = 2;  
f(x) + y;
```

Сборщик мусора

- Постраничное управление памятью
 - Память освобождается при смене страницы
- Подсчёт ссылок
 - Каждая область памяти ассоциирована с числом ссылок на неё
 - Число пересчитывается при смене указателя
 - Память очищается когда число становится равным 0
- Пометить и смести (mark-and-sweep)
 - GC помечает память
 - Собирает и вычищает неиспользуемую память

Замыкания

- Возвращение функции из вызова функции

```
function f(x) {  
  var y = x;  
  return function (z) {  
    y += z;  
    return y;  
  }  
}  
  
var h = f(5);  
h(3);
```

- Можно использовать для создания “private” полей
 - <http://www.crockford.com/JavaScript/private.html>

Исключения

- **Выкидываем исключение**

```
throw "Error2";
```

```
throw 42;
```

```
throw {toString: function() { return "I'm an object!"; } };
```

- **ЛОВИМ**

```
try {
```

```
  } catch (e if e == "FirstException") {    // do something
```

```
  } catch (e if e == "SecondException") {  // do something else
```

```
  } catch (e){                            // executed if no match above
```

```
}
```

Особенности объектов

- Динамическое расширение
 - Значения свойств определяются в момент выполнения (run-time)
- Инкапсуляция
 - Объекты могут содержать управляющие конструкции и приватные данные
- Полиморфизм
 - Одни объекты могут быть использованы вместо других
- Наследование
 - Через прототипы

Многопоточность (Concurrency)

- Сам по себе JavaScript однопоточный
- AJAX предоставляет возможность конкуренции
 - Создаём XMLHttpRequest и устанавливаем функцию обратного вызова
 - Исполняем запрос и дальше всё работает асинхронно
 - Ответ от удалённого сервера вызывает функцию обратного вызова
 - Событие ждёт своей очереди в общем цикле исполнения
- Другая форма многопоточности
 - Использование setTimeout для исполнения нескольких задач

eval

- Вычисляет строку кода
 - Строка JavaScript кода вычисляется в пространстве имён вызывающего кода
- Пример

```
var code = "var a = 1";  
eval(code); // а теперь '1'  
var obj = new Object();  
obj.eval(code); // obj.a теперь 1
```
- Наиболее частое применение
 - Быстро обрабатывает JSON полученный через AJAX
- Чего стоит иметь eval в языке
 - Представьте eval в C. Как его можно реализовать?

Необычные особенности

- Встроенные функции
 - eval, проверка типа во время исполнения
- Регулярные выражения
 - Поддержка синтаксисом языка
- Динамическое расширение объектов
- Интегрирование по методам в объекте
 - for (variable in object) { statements }
- Выражение with
 - with (object) { statements }

Ресурсы

- Brendan Eich, slides from ICFP conference talk
 - www.mozilla.org/js/language/ICFP-Keynote.ppt
- Tutorial
 - <http://www.w3schools.com/js/> (still there?)
- JavaScript 1.5 Guide
 - http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Guide
- Douglas Crockford site
 - <http://www.crockford.com/JavaScript/>
 - <http://20bits.com/2007/03/08/the-philosophy-of-JavaScript/>